TracenPoche ...

la géométrie dynamique pour tous.

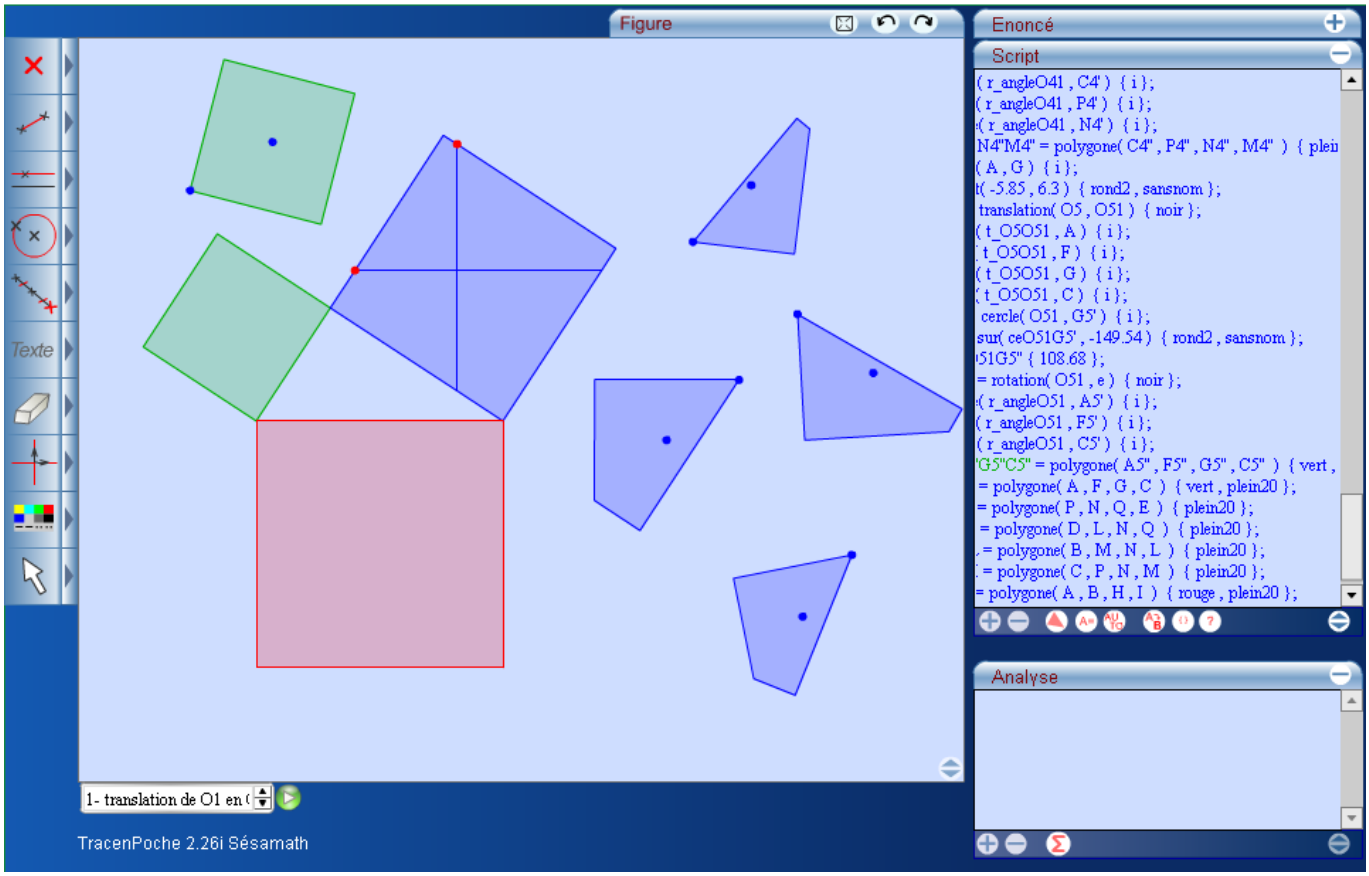


Table des matières

1. Introduction

TracenPoche est le logiciel de géométrie dynamique de MathenPoche. C'est un des outils de MathenPoche. Ce dernier étant développé dans l'environnement Flash © Macromedia, fort logiquement, cette plate-forme a été choisie pour le développement de TeP.

Ce choix n'est pas sans conséquence. Action Script, le langage à vocation internet utilisé dans Flash, n'est pas aussi généraliste que les langages comme le C ou le Pascal par exemple. En outre, c'est un langage script, ce qui signifie que le programme nécessite un lecteur pour pouvoir être exécuté. Dans notre cas, il s'agit du Lecteur Flash (disponible gratuitement) ou d'un explorateur Internet utilisant le plugin adéquat.

Toutefois, le choix de cette plate-forme s'avère très satisfaisant dans de nombreux domaines, en commençant par celui de l'affichage d'objets sur la scène, ce qui est quand même très utile en géométrie dynamique. En outre, TeP2 est développé en Action-Script 2.0, qui permet le typage des variables, donc le développement d'un code plus clair, plus structuré et plus rigoureux. Ceci, ajouté à une optimisation continuelle du compilateur, apporte une vitesse d'exécution accrue et une meilleure stabilité du programme.

Néanmoins, Action-Script 2.0 se prête très mal à l'enregistrement de fichiers, ce qui est légitime pour internet mais très pénible en local. Nous avons donc développé des programmes spécifiques pour contourner ces problèmes.

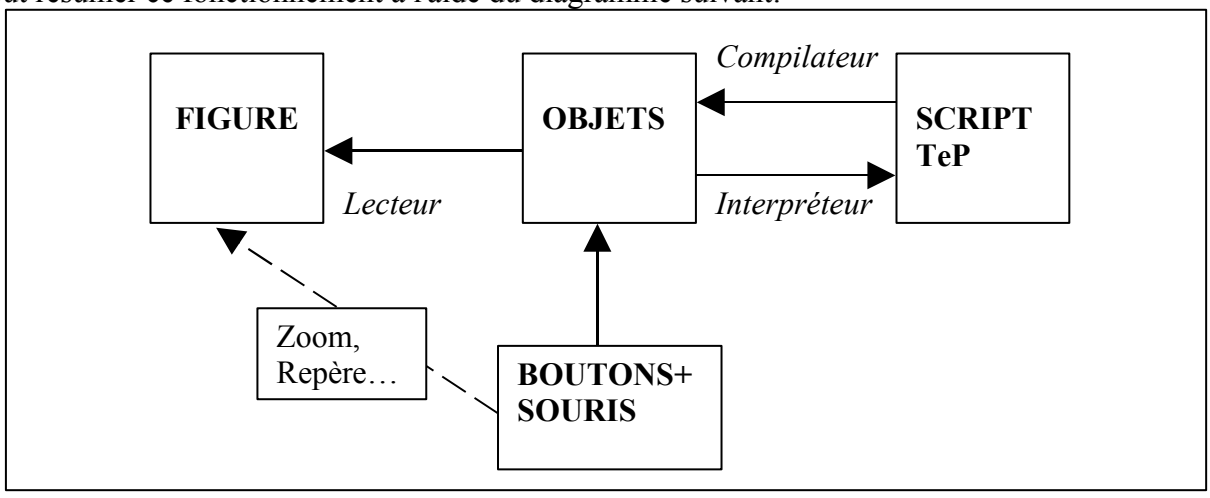
Structure du programme

Tous les éléments géométriques de la figure (ou presque), ainsi que les variables, sont des objets codés informatiquement. La figure est construite directement à partir de ces objets.

Pour construire une figure, on dispose de deux méthodes complémentaires.

- ✓ **Le script**
  Un script TeP est un fichier texte. Un script permet la définition des objets. Ce script n'agit pas sur la figure. Il agit sur la définition des objets, et ce sont ces objets qui permettent l'affichage des éléments.
- ✓ **Les boutons**
  Les boutons permettent également la définition directe des objets. Dans le cas de la création d'un élément à l'aide de la souris, un objet est alors créé, le script est mis à jour, et enfin l'élément est affiché.

On peut résumer ce fonctionnement à l'aide du diagramme suivant:



De la même manière, quand on déplace un objet de la figure, on modifie en réalité l'objet sous-jacent, ce qui permet alors une actualisation du script et donc de la figure.

Finalement, pour un objet, on distingue trois éléments :
- l'objet réel : il est invisible et est codé informatiquement.
- le dessin de l'objet à l'écran : représentation de l'objet.
- le script : représentation textuelle de l'objet.

Le script

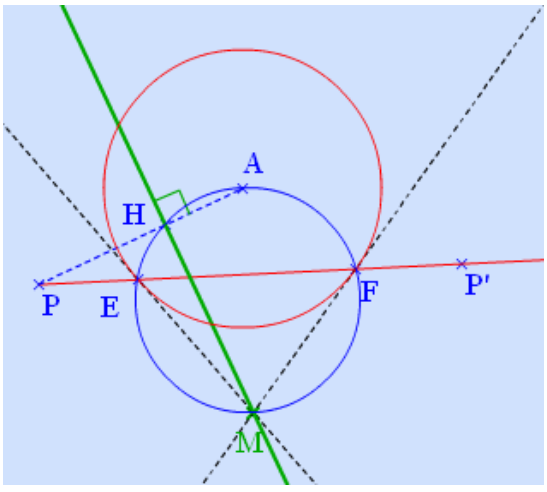
Le script contient des commandes dont la syntaxe est presque toujours la suivante :
```
nom_objet = nature_objet(paramètres de l'objet) {options d'affichage de cet objet}
```

Pour certains objets, comme les segments et les droites, d'autres syntaxes sont possibles. En effet, il y a plusieurs compilateurs/interpréteurs définis dans TeP. Mais dans la plupart des cas,
parmi les nombreuses syntaxes possibles, c'est celle-ci qui a été choisie comme syntaxe par défaut.

De nombreuses raisons nous ont poussés à faire ce choix.

Tout d'abord, l'expérience montre qu'elle est naturelle pour l'élève.
Elle oblige l'élève à définir clairement ses objets. La syntaxe proposée aide l'élève dans ce travail, en ce sens qu'elle propose un canevas pour répondre successivement aux questions: *"Qu'est-ce que je veux construire ? Quel est son nom ? Quelles sont ses propriétés ? "* Et enfin, et cela seul est facultatif, *comment doit être l'apparence de mon objet ?* Elle a également le mérite d'imposer une rigueur à l'élève en ce sens que le protocole de définition doit être rigoureusement suivi. L'implicite n'est pas autorisée. Cette démarche est volontaire. Elle nous semble saine à une époque où l'élève a trop souvent tendance à négliger l'implicite et à se contenter d'explications vagues et incomplètes.

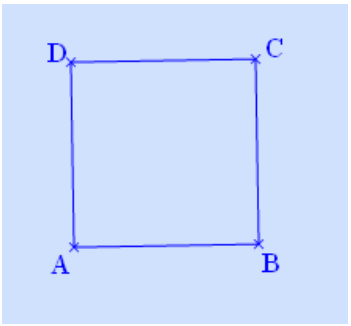
Le script a également l'avantage de permettre une compréhension parfaite de la construction d'une figure complexe.

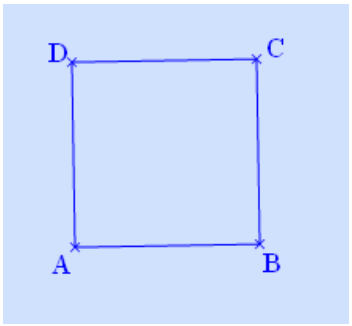


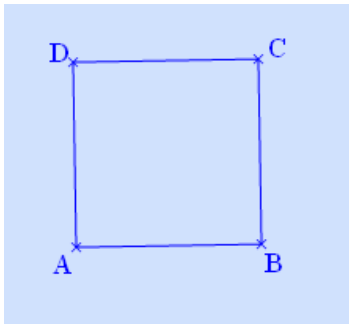
Cette figure montre une construction de la polaire (HM) du point P par rapport au cercle de centre A.
Quand on ne connaît pas la construction d'une polaire, il n'est pas évident de comprendre les mécanismes de construction de celle-ci.
Avec le script, cela ne pose pas de problème.

Même en ce qui concerne les figures simples, le script a son importance.

Considérons par exemple les figures suivantes :

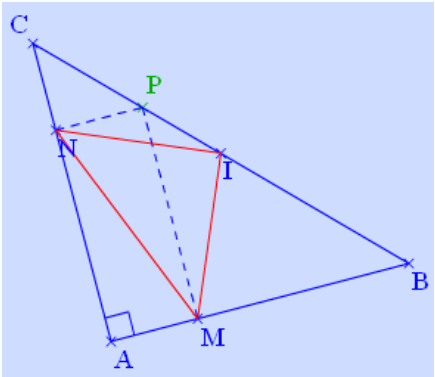






Ce sont des carrés visiblement ! Les figures sont identiques !
Oui, mais ces carrés sont construits différemment.
Dans la figure 1, le carré est construit à partir du segment [AB] et deux rotations.
Dans la figure 2, le carré est construit à partir de son centre O (non visible), de A et de trois rotations.
Dans la figure 3, le carré est construit à partir du segment [AC], d'une médiatrice et d'un cercle de centre O (non visibles).
Donc leurs scripts sont nettement différents.
En outre, si on saisit le point A dans la figure 1, le carré pivote en suivant le segment [AB].
Par contre, dans la figure 2, il tourne autour de O (par similitude).
Les scripts permettent de lever clairement ces ambiguïtés.

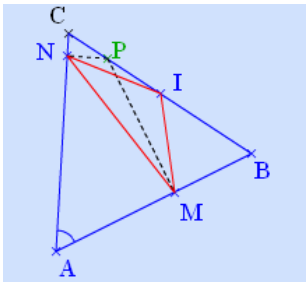
Dans l'exemple suivant, la figure est claire :



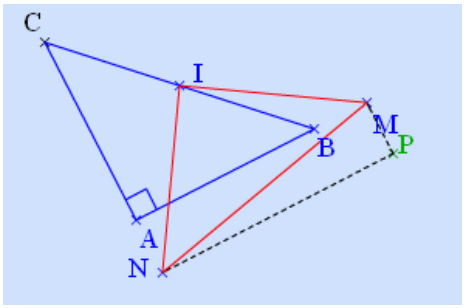
ABC est un triangle rectangle isocèle. I est le milieu de [BC] (c'est immédiat quand on déplace les points B ou C).
P est mobile mais ne se déplace que sur [BC].
M et N sont clairement les projetés orthogonaux de P sur les segments [AB] et [AC].

La couleur rouge indique clairement que l'on s'intéresse ici à la nature du triangle ABC.
Dans ce cas, il semble que MIN soit rectangle isocèle.
Il est peut être utile d'étudier la nature de MIN, dans d'autres cas de figure : position de I, nature de ABC etc…(c'est une tendance inhérente aux "matheux" de tout vouloir généraliser…).

La redéfinition des objets s'avère souvent très délicate dans la plupart des logiciels.
Le script, non pas seulement descriptif, mais dynamique, permet de redéfinir les objets sans aucune difficulté, puisqu'il suffira de modifier un ligne ou deux pour obtenir la nouvelle configuration.
Par exemple, la modification d'une seule ligne, on obtient la figure suivante :



Une conjecture s'impose immédiatement…
On peut aussi se demander ce qui se passe si le point P appartient à la droite (BC) ...

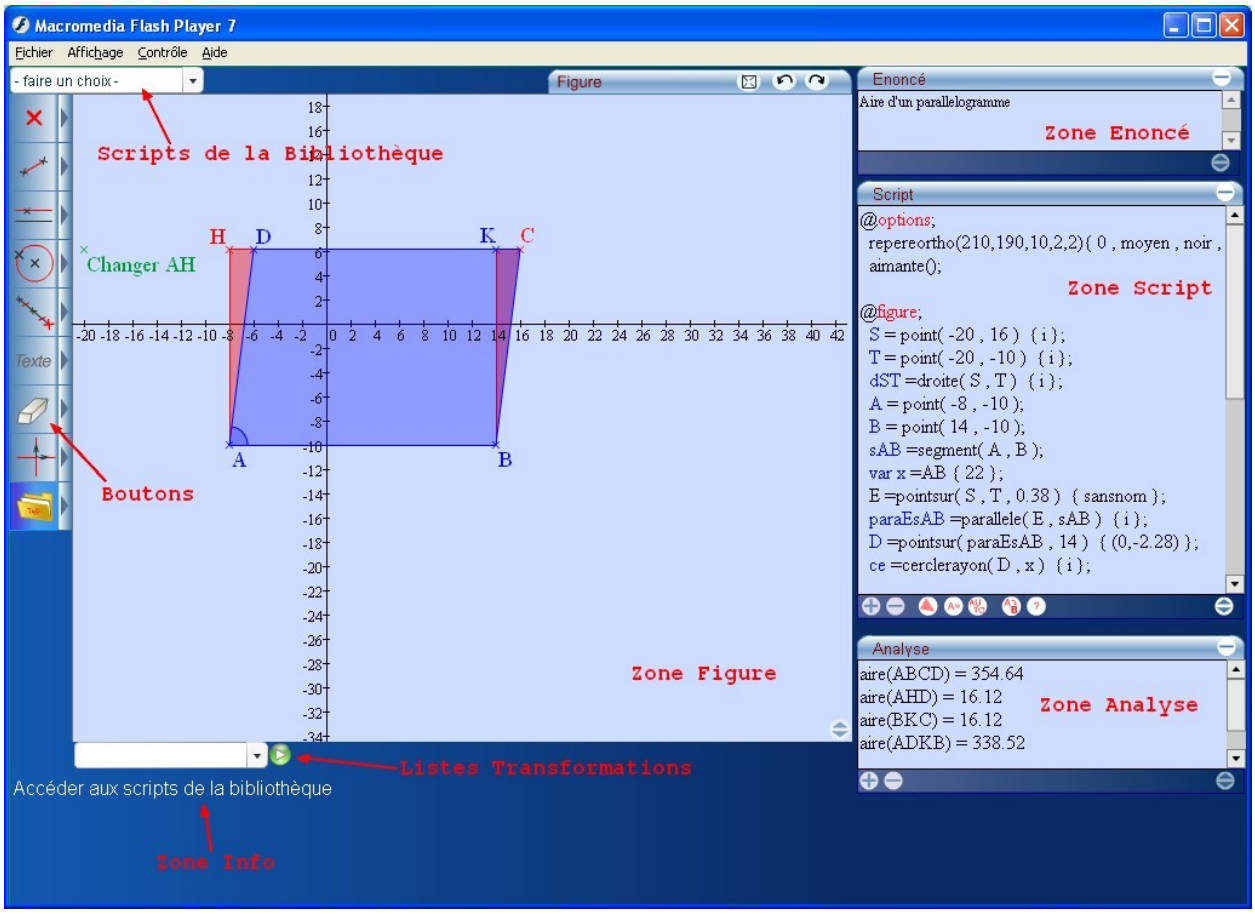


... et on constate que le résultat semble toujours vrai.

2. Présentation générale

2.1 Fonctionnement du programme

2.1.1 Interface



La **zone de travail** comporte un **script**, une **figure**, un **énoncé**, une **zone d'analyse**, une **boîte liste** permettant d'utiliser des transformations, une **boîte liste** permettant de charger des figures, un **texte d'informations** et enfin des **boutons**.

La **zone info** affiche un message qui varie selon l'action en cours.
Cette zone constitue une aide appréciable quand on découvre le logiciel.

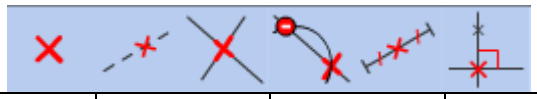
La **boîte liste fichiers** apparaît quand on clique sur le bouton "Fichiers".
Cette boîte contient tous les fichiers textes (contenant les scripts) proposés dans le répertoire du programme(par défaut).

Les **boutons** situés au bas de la zone script sont décrits dans la section zone script.

La zone **énoncé** n'est pas modifiable. Elle est remplie au moment du chargement du fichier.

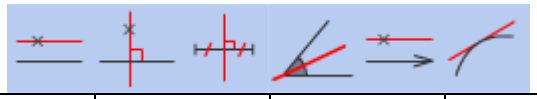
2.1.2 Boutons et clavier

Il y a 10 lignes de boutons.
Les six premières lignes contiennent les boutons de construction .
Les trois dernières contiennent des boutons de commandes comme par exemple les zooms, le déplacement de la figure, la mise en forme d'un objet…
La dernière permet d'annuler l'action en cours.

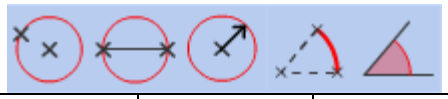


1	2	3	4	5	6
Créer un point	Créer un point sur un objet	Point d'intersection	Point d'intersection évitant un point existant	Milieu	Projeté orthogonal

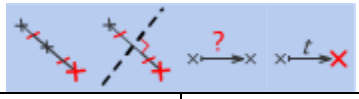
1	2	3	4	5
Créer un segment	Créer une droite	Créer une demi-droite	Créer un polygone	Créer un vecteur



1	2	3	4	5	6
Créer une parallèle	Créer une perpendiculaire	Créer une médiatrice	Créer une bissectrice	Créer une droite définie par un vecteur directeur	Créer une tangente

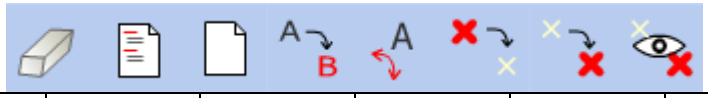


1	2	3	4	5
Cercle défini par centre et un point	Cercle défini par un diamètre	Cercle défini par centre et rayon	Construire un arc	Marquer un angle

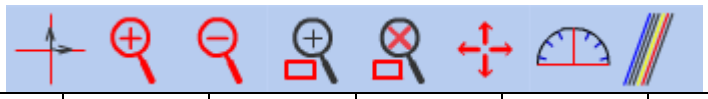


1	2	3	4
Symétrique par rapport à un point	Symétrique par rapport à une droite	Déclarer une transformation	Image par une transformation

1	2	3	4	5	6	7
Afficher un texte	Mesurer une distance	Mesurer un angle	Définir une variable	Animer une construction	Activer ou Désactiver la trace d'un objet	Construire un lieu de points



1	2	3	4	5	6	7	8
Supprimer un objet	Effacer et recharger la figure de base	Effacer toute la figure	Renommer un objet	Bouger le nom d'un point	Rendre invisible	Rendre visible	Voir/cacher un objet invisible



1	2	3	4	5	6	7	8
Afficher ou masquer le repère	Zoom +	Zoom –	Zoom sur zone	Annuler zoom sur zone	Déplacer la figure	Afficher le rapporteur	Afficher le guide-âne

1	2	3	4	5	6	7	8	9
Couleurs et styles	Sauver la figure dans MeP	Aide	Ouvrir scripts	Fiche complète de la figure	Fiche complète de la figure pour insertion dans OOo	Sauver fiche de la figure	Sauver fiche de la figure OOo	Options générales

Annuler l'action en cours

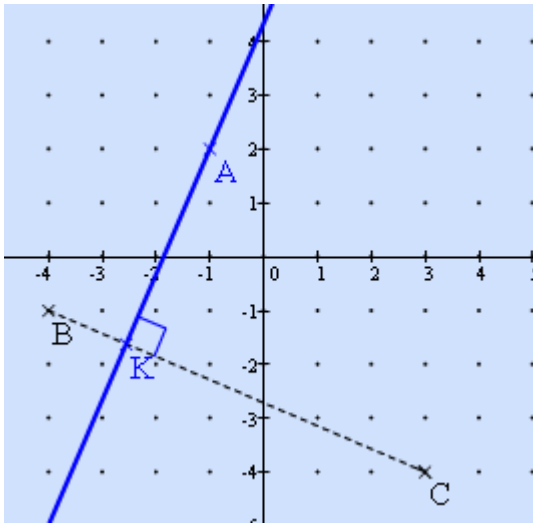
2.1.3 La structure du script

La ***zone script*** contient le script de la figure.
Un script est divisé en deux sections :

```
@options ;
@figure ;
```

Ces mots clefs sont obligatoires (comme le caractère @ et le point-virgule).
D'ailleurs quand on affiche toute la figure à l'aide du bouton, ils apparaissent systématiquement dans le script.

Voici un exemple de script et sa figure correspondante :




```
@options;
  repereortho(216,175,27,1,1){ 0 , moyen ,
noir , num1 };
  grille();
  aimante();

@figure;
  A = point( -1 , 2 );
  B = point( -4 , -1 )  { noir };
  C = point( 3 , -4 )  { noir };
  s =segment( B , C )  { noir , 7 };
  hA =perpendiculaire( A , s )  { 2 };
  K =intersection( hA , s );
```

a) Les options

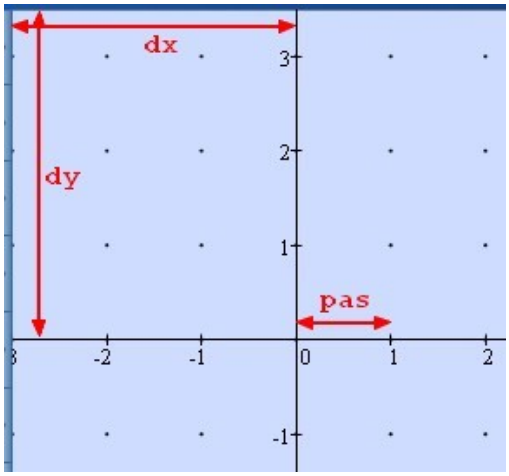
La section `@options;` permet de préciser certaines options qui concernent le repérage :
nature et format du repère, trame et aimantage.

Les repères
Le repère est dans la section `@options;` ce n'est pas un objet.

Deux types de repère sont possibles :
- repère **orthonormé**
- repère **orthogonal**

Voici la syntaxe de la commande **repère orthonormé** :

```
repereortho(dx,dy,pas,gradux,graduy);
```



avec

dx, **dy** et **pas** sont en pixels (points physiques de l'écran ; par défaut, le repère est centré)
pas indique la longueur en pixels d'unité.
Une fois l'unité choisie, on choisit la graduation.
Par exemple si **gradux** = 2, on gradue l'axe des abscisses toutes les deux unités.

Cette syntaxe est donc relativement complexe.

Mais on peut utiliser la commande **repereortho(xmin,xmax,ymin,ymax);**

On indique à **TeP** les abscisses et ordonnées maximales et minimales que l'on souhaite voir apparaître.

Et **TeP** calculera lui-même les paramètres permettant d'avoir un repère orthonormé approchant le plus celui demandé.

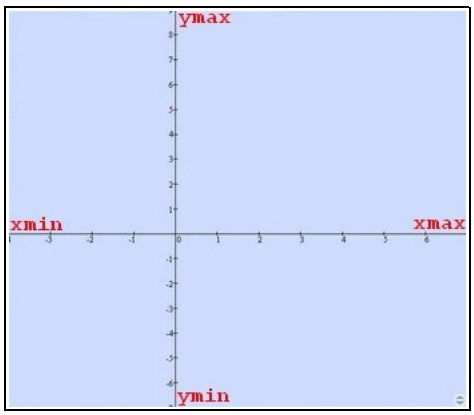
Dans **TeP**, il y a toujours un repère, même si aucun repère n'est défini par l'utilisateur.
Si la zone de la figure a pour dimensions 400 x 300 (pixels) alors, il s'agit du repère suivant :

```
repereortho(200,150,10,1,1) ;
```

Voici la syntaxe de la commande **repère orthogonal**:

```
repere(xmin,xmax,ymin,ymax,gradux,graduy) ;
```

qui permet de construire le repère :



La trame

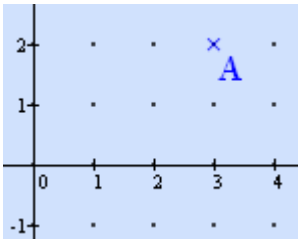
L'option **trame();** affiche un quadrillage à l'écran adapté aux graduations des axes.

L'option **grille();** affiche une grille à l'écran adaptée aux graduations des axes.

Ces 2 options s'appliquent aux graduations principales même si celles-ci ne sont pas dessinées comme certaines options le permettent (voir **3.1 Les options**).

Aimantage des points

L'option `aimante();` est l'option complémentaire : elle permet de déplacer les points libres uniquement sur les nœuds la grille.




```
@options;
  repereortho(310,270,30,1,1){ 0 , moyen , noir , num1 };
  grille();
  aimante();

@figure;
  A = point( 3 , 2 );
```

Le point A se déplacera sur les points de coordonnées entières.

b) La syntaxe générale

La section `@figure;` contient la description textuelle de la figure.
La syntaxe de base est la suivante :

```
Nom_objet = nature_objet(paramètres) { option1, option2,…} ;
```

Voici quelques **commandes** simples :

`A = point(xA,yA) ;`	Construit un point A de coordonnées (xA, yA). Si le repère n'est pas défini, alors ce point est construit dans le repère par défaut.
`M = pointsur(A,B,x) ;`	M d'abscisse x de la droite (AB), dans le repère (A, AB)
`E = intersection(o1,o2) ;`	E point d'intersection des objets o1 et o2 o1 et o2 sont des objets (demi-droite, droite, segment ou cercle)
`s = segment(A,B) ;` ou `s = [AB] ;`	Construit le segment [AB]. Ce segment est appelé s.
`d = droite(A,B) ;` ou `d=(AB) ;`	Construit la droite (AB). Cette droite est référencée par son nom d
`C = cercle(A,B) ;`	Construit le cercle de centre A et passant par B

Les **options** sont facultatives.
Elles permettent de :

- définir la couleur de l'objet
- définir le style du trait (épaisseur si le trait est continu ou type de pointillés)
- masquer un objet
- préciser la position du nom du point
- modifier l'aspect des points
- coder les segments
- régler l'opacité des surfaces coloriées
- animer des points
- garder la trace d'un objet
- préciser la taille de la police d'un texte

- afficher les coordonnées d'un point

L'ordre des options est sans importance.

→ *Voir la partie référence pour* *le détail des options page 55*

Exemple :
La commande `A = point(1,3.6)  {rouge,3,trace,(-2,-3.3)}`

a pour résultat (après déplacement du point) :

c) *Boutons de la zone script*

Au bas de la zone script, on distingue plusieurs boutons :

1 **Modifier la taille des caractères dans la zone script** | **2** **Actualiser la figure** | **3** **Actualiser le texte** | **4** **Désactive et active la réactualisation automatique du script** | **5** **Renommer un objet** | **6** **Insérer des accolades** | **7** **Aide**

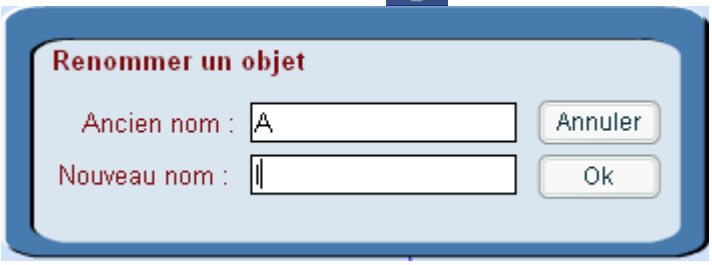
1 : Les boutons  permettent d'agrandir ou de réduire la taille des caractères dans la zone de script.

2 : Quand on entre un texte dans cette zone, la figure n'est pas reconstruite automatiquement. D'où le bouton  . On peut également appuyer sur la touche F9 du clavier.

3 et 4 : Lorsque l'on déplace un objet à la souris, le script se met à jour une fois que la souris a été "lâchée". Pour des scripts longs cela nécessite parfois un peu de temps durant lequel le logiciel semble ne plus répondre. Pour désactiver cette réactualisation du script il suffit de cliquer sur le bouton  .

Si on veut actualiser le script il suffit ensuite d'appuyer sur le bouton  .

5 : Supposons qu'un point A soit défini au début du script et supposons qu'il soit utilisé de nombreuses fois dans le script. Si nous voulons changer le nom de ce point en I, il faut changer tous les "A" en "I" : fastidieux et source d'erreurs. En cliquant sur le bouton  , une boite de dialogue apparaît :



En validant, tous ces changements sont effectués automatiquement.

6 : Le bouton  permet d'insérer rapidement des accolades dans le script afin de définir les options d'un objet.

7 : Le bouton  permet d'accéder à l'aide relative au mot sous lequel est situé le curseur dans la zone script.

2.1.4 Structure de la zone analyse

La ***zone analyse***, comme son nom l'indique, permet d'analyser la figure.
Les éléments calculés, les réponses aux questions, ne sont pas des objets. Ils ne sont donc pas utilisables directement par le script. Cette zone fournit des informations sur la figure mais il n'y a pas de lien dans le sens zone analyse → zone script.
Si un calcul est nécessaire pour construire une figure alors il faut utiliser la commande **`var.`**
Par exemple **`var x = AB ;`** définit une variable **`x`** égale à AB. Cette variable peut-être utilisée dans les calculs de la zone analyse.

Les commandes de zone analyse :

Distance de deux points	**AB** =	L'unité est celle du repère orthonormé.
Angle	**angle**(BAD)=	Valeur en degrés de l'angle géométrique $\widehat{BAD}$. Il n'y pas d'orientation de l'angle.
Anglev	**anglev**(BAD)=	Valeur en degrés de l'angle $\widehat{BAD}$. Renvoie une valeur entre 0° et 360°
Aire	**aire**(ABCDE)= ou **aire**(c)=	Aire d'un polygone, d'un cercle ou d'un secteur délimité par un arc.
Périmètre	**périmètre**(ABC)= ou **périmètre**(c)=	Périmètre d'un polygone, d'un cercle ou longueur d'un arc.
Longueur	**longueur**(ABC)=	Longueur de la ligne polygonale définie par une liste de points.
Calcul	**calc**(expression)=	Valeur de l'expression.
Equation d'une droite	**er**(AB):	Équation réduite de la droite (AB). Si l'option aimante() est activée, alors la programme affiche le résultat exact et réduit de l'équation.
Equation d'une droite	**ec**(AB):	Équation cartésienne de la droite (AB). Si l'option aimante() est activée, alors la programme affiche le résultat exact et réduit de l'équation.
Nature d'un triangle	**nature**(ABC)=	Quelconque, isocèle, équilatéral, rectangle, rectangle isocèle.
Nature d'un quadrilatère	**nature**(ABCD)=	Quelconque, trapèze, parallélogramme, losange, rectangle, carré.
Alignement	A,B,C **alignés ?**	Oui ou non.
Position relative de deux droites	**position**(AB,CD)=	Sécantes, parallèles, perpendiculaires.
Création d'un variable	**sto** x	Création d'une variable x égale au résultat de la ligne précédente.
PGCD de 2 nombres	**pgcd**(x,y)=	Calcule le PGCD de x et y.
PPCM de 2 nombres	**ppcm**(x,y)=	Calcule le PPCM de x et y.

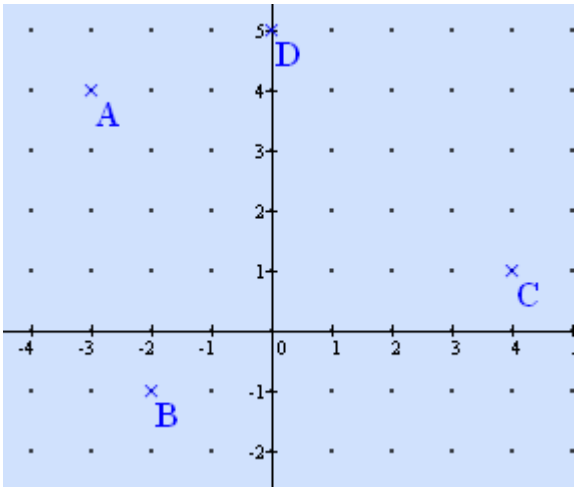
Simplification d'une fraction	**simplifie**(x,y)=	Simplifie la fraction x/y.
Forme rationnelle d'un nombre	**exact**(x)=	Renvoie la forme rationnelle du nombre x.
Dimensions d'un triangle	**dim**(ABC)=	Renvoie les dimensions du triangle ABC.
Abscisse d'un point	**abscisse**(A)=	Renvoie l'abscisse du point A
Ordonnée d'un point	**ordonnee**(A)=	Renvoie l'ordonnée du point A
Coordonnées d'un point	**coord**(A)=	Renvoie les coordonnées du point A

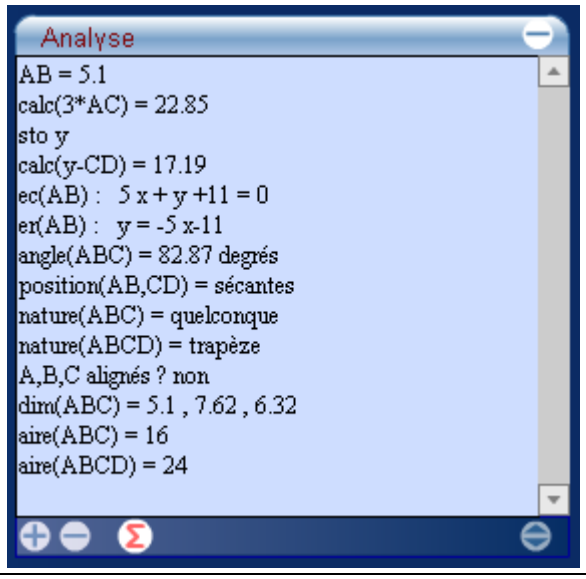
La commande **calc** accepte les commandes **angle** et **aire**, ainsi que les variables créées grâce à la commande **sto**.
Par exemple :

calc(angle(ABC)+angle(ACB))=

L'évaluation d'une expression à l'intérieur de la **zone analyse** se fait après l'appui sur la touche F9 ou sur le bouton  se trouvant en bas de celle-ci.

Voici un exemple d'utilisation de la **zone analyse** :





2.1.5 Boîte de dialogue transformations

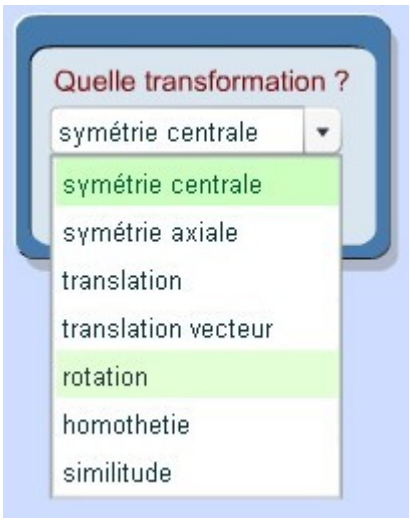
TeP permet de définir des transformations comme objets (non visuels). Elles sont ainsi réutilisables à volonté pour construire plusieurs images.

Quand une transformation a été définie (par le script ou à la souris), elle apparaît alors dans le boîte liste Transformations.
Quand on clique ensuite sur le bouton " image d'un point par une transformation ", le programme demande d'abord de cliquer sur le point dont on veut construire l'image puis ensuite de choisir la transformation. On sélectionne la transformation à l'aide de cette boîte.

Pour déclarer une transformation à la souris, il faut cliquer sur le bouton "déclarer une transformation".

Définir une transformation :

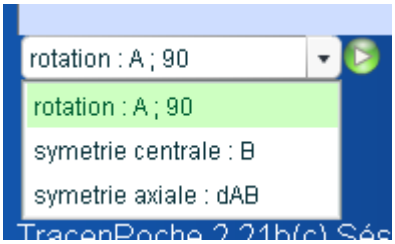
On choisit la transformation



Après avoir défini ses caractéristiques, on obtient

Utiliser une transformation :

On sélectionne la transformation à utiliser



On valide son choix en cliquant sur le bouton vert

2.1.6 Premier exemple : construction d'un carré

Plusieurs méthodes sont possibles pour construire un carré (voir **Introduction**).
Dans cet exemple, on choisit de construire ce carré à partir d'un segment [AB] et de deux perpendiculaires passant par A et B.

On commence par construire deux points A et B.

Placer le point A à la souris.

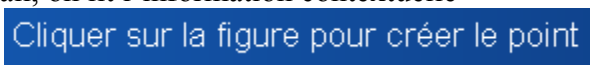
☞ Cliquer sur le bouton point : 

Le fond du bouton change de couleur : 

Ce qui indique qu'une construction est en cours.
Quand une construction est en cours, vous ne pouvez pas en démarrer une autre tant qu'elle n'est pas terminée.

Pour **annuler** la construction de ce point, il suffit de cliquer à nouveau sur le bouton qui reprend alors son aspect normal. Cette technique est valable pour la construction de tous les objets.

Au bas de l'espace de travail, on lit l'information contextuelle



Le programme affiche alors une zone de texte et propose un nom par défaut.
Si le nom convient, taper **ENTREE** (ou cliquer avec la souris sur la zone de dessin) sinon, entrer un autre nom et taper **ENTREE.**
Remarquer que le fond du bouton a changé et reste plus foncé que les autres.
Ce fond indique quel était le type du dernier objet construit.
Lors de la création d'un autre point, on disposera de deux méthodes pour le créer :
– soit on clique à nouveau sur le bouton
– soit on utilise la combinaison de touches **CTRL + MAJ** qui équivaut à cliquer à nouveau sur ce bouton.

Le script est devenu ceci :

```
@options;
@figure;
  A = point(-5,2) ;
```

Des coordonnées s'affichent. Mais dans quel repère ?

Toute figure se construit dans un repère qui par défaut est orthonormé.

☞ *Pour l'afficher, cliquer sur le bouton* 

Un repère s'affiche. On constate que les coordonnées de A correspondent aux coordonnées écrites dans le script.

Le script a été mis à jour en conséquence :

```
@options;
  repereortho(310,270,30,1,1){ 0 , moyen , noir , num1 };
@figure;
  A = point(-5,2);
```

Pour masquer ce repère, il suffit de cliquer à nouveau sur le bouton.

Ajouter le point B et le segment [AB]

☞ *Créer un point B.*
☞ Pour *créer le segment [AB],* à l'aide du bouton *créer le segment à l'aide deux points*, on peut utiliser la souris : dans ce cas le script donne automatiquement un nom au segment : sAB =segment(A,B);
Ce nom défini automatiquement par le programme suit la syntaxe : s suivi du nom du 1er point suivi du nom du 2ème point.
Cet ordre a une importance, comme par exemple quand on définit un "*pointsur*" ce segment.

☞ *Déplacer le segment [AB].*

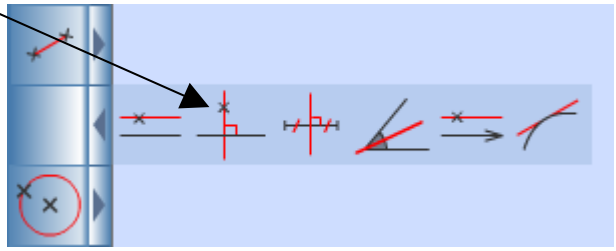
En rapprochant la souris de ce segment, le curseur change de forme. Ce qui montre qu'un objet est proche du curseur. En appuyant sur le bouton, le curseur change à nouveau et prend la forme d'une main saisissant le segment. Tout déplacement de la souris provoque alors le déplacement (par translation) du segment.

Remarque importante : choix d'un objet parmi plusieurs sous la souris.
Si la souris est proche de plusieurs objets, le programme choisira de lui même un objet proche du curseur (il sélectionnera en priorité un point, un segment, une droite).
Rapprocher le curseur souris du point A (légèrement sur le segment). Il y a deux objets à côté du curseur. Pour passer en revue tous les objets proches, il suffit, pendant que le bouton de la souris est enfoncé, de frapper la barre espace : tous les objets proches défilent (boucle circulaire).
On peut donc déplacer n'importe quel objet proche du curseur.

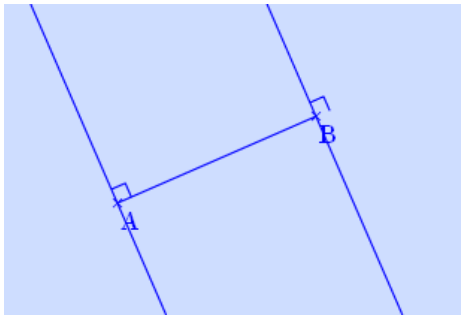
Construire les perpendiculaires au segment [AB] passant par A et B .

Les objets droite (parallèle, perpendiculaire, médiatrice, bissectrice, de direction un vecteur, tangente) sont ceux de la 3ème ligne.

Utiliser donc le bouton :



Il suffit de suivre les indications dans la zone information.
On obtient :



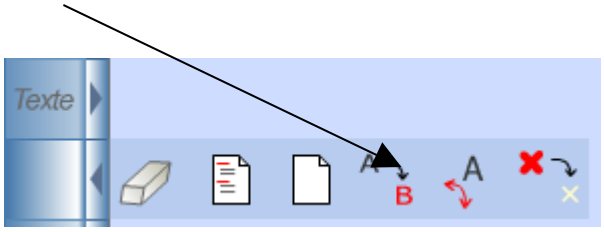
Et le script correspondant :

```
@options;
  repereortho(310,270,30,1,1){ 0 , moyen , noir , num1 ,i};

@figure;
  A = point( -0.23 , -0.33 );
  B = point( 4.2 , 1.6 );
  sAB =segment( A , B );
  perpAsAB =perpendiculaire( A , sAB );
  perpBsAB =perpendiculaire( B , sAB );
```

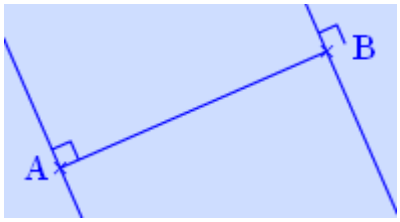
Ces perpendiculaires sont automatiquement nommées par le programme. La syntaxe choisie est suffisamment parlante. Nous reviendrons sur ce point plus loin.

Repositionner le nom des points A et B.

☞ *Utiliser le bouton (bouger le nom d'un point)*



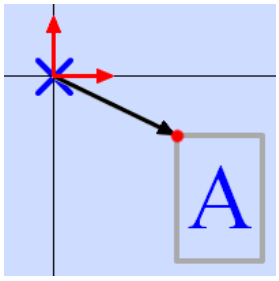
Et le bouton se met en sur-brillance.
Nous sommes donc dans le mode déplacement des noms des points.
Quand les noms sont positionnés correctement , il faut cliquer à nouveau sur le bouton pour sortir de
ce mode.



Le script a changé :

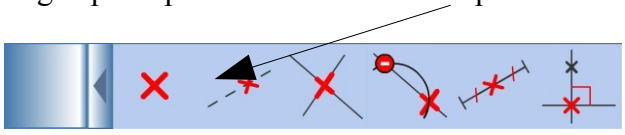
```
A = point( -0.23 , -0.33 )  { (-0.67,-0.37) };
B = point( 4.2 , 1.6 )  { (0.37,-0.43) };
```

Ces coordonnées apparues entre accolades sont celles du point I (invisible) situé en haut à gauche de la zone couverte par le texte : voir le point rouge sur le schéma ci-dessous. Par défaut, elles valent (0 ;0) et dans ce cas, le point rouge se situe exactement sur la marque du point A (sur la croix).
Les coordonnées (-0.67,-0.37) sont celles du vecteur d'origine "la croix" et d'extrémité le point rouge, dans un repère invisible ici.



Construire le point D.

Le deuxième bouton de la ligne point permet de construire un point sur un objet.



En **zone info**, on lit "cliquer sur un objet ou sur l'origine d'un axe".

☞ *Cliquer sur la droite perpAsAB.*

☞ *Appeler ce point D.*

On peut le déplacer mais il restera sur l'axe.

Voici la syntaxe de ce type de point :

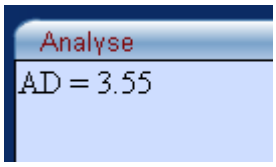
```
Nom_point  =pointsur(objet, x) ;
```

x permet de positionner précisément ce point sur l'objet.
Ici, on a :

```
D =pointsur( perpAsAB , 3.55 );
```

Que signifie 3.55 ?

Pour le savoir, dans la fenêtre analyse taper **AD=**, et actualiser *(touche **F9** ou bouton*  *)* :



Il s'agit donc de la distance AD, toujours dans le repère par défaut.
Si D passe de l'autre côté de A, alors cette position devient négative. L'orientation de l'axe est définie par le programme une fois pour toute.

Pour un carré, il nous faut AD = AB. Il est vrai qu'il aurait suffit de construire un cercle mais ce sont les outils utilisés qui importent ici.

☞ *Insérez la commande* **`var x =AB;`** *avant la déclaration de D et actualisez la figure.*

```
A = point( -0.23 , -0.33 )  { (-0.67,-0.37) };
B = point( 4.2 , 1.6 )  { (0.37,-0.43) };
sAB =segment( A , B );
perpAsAB =perpendiculaire( A , sAB );
perpBsAB =perpendiculaire( B , sAB );
var x =AB { 4.832163076718... };
D =pointsur( perpAsAB , 3.55 );
```

le script affiche entre accolades la longueur de AB donc la valeur actuelle de x

Cette commande définit une variable nommée ***x*** et qui est égale à la longueur AB.
La commande **`pointsur`** admet des variables comme paramètre :

☞ R*emplacer 3.55 par x.*

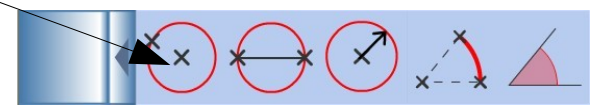
```
D =pointsur( perpAsAB , x );
```

☞ *Actualiser la figure.*

Les distances AB et AD sont maintenant égales. Le point D n'est plus libre sur l'axe mais sa position dépend de celles de A et de B.
Pour bouger le point D, il faut donc, par exemple, bouger le point B.

Construction de C

☞ *A l'aide du bouton construire le cercle de centre B et passant par A.*



☞ *A l'aide du bouton*  *construire les points d'intersection du cercle et de la perpendiculaire en B.*
Le programme propose un nom pour une des deux intersections. Il s'agit de C.
Automatiquement, il crée la seconde intersection, nommée C2.

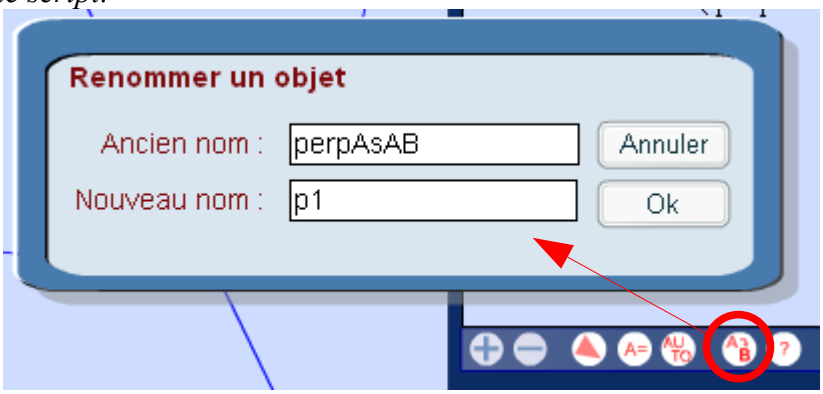
```
A = point( -0.23 , -0.33 )  { (-0.67,-0.37) };
B = point( 4.2 , 1.6 )  { (0.37,-0.43) };
sAB = segment( A , B );
perpAsAB = perpendiculaire( A , sAB );
perpBsAB = perpendiculaire( B , sAB );
var x =AB { 4.83216307671834 };
D = pointsur( perpAsAB , x );
ceBA = cercle( B , A );
C2 = intersection( ceBA , perpBsAB , 1 );
C = intersection( ceBA , perpBsAB , 2 );
```

Le point C ne nous intéresse pas.
Il faut supprimer ce point :

- soit on supprime la ligne correspondante du script (sans oublier d'actualiser la figure ensuite),
- soit on utilise la gomme pour supprimer le point (quand plusieurs objets sont proches de la gomme, la gomme efface en priorité les points, en sélectionnant le point le plus proche de la souris).

☞ *Pour renommer le point C2 en C, le plus simple est de travailler sur le script.*

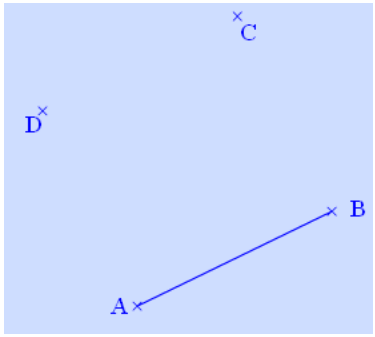
☞ *Pour renommer perpAsAB en p1, c'est plus délicat, car il faut remplacer tous les perpAsAB par p1 dans tout le script.*



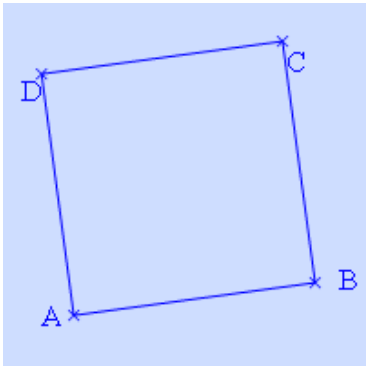
Le cercle et les perpendiculaires sont inutiles. Pour les rendre invisible, il suffit de rajouter aux options de ces objets le paramètre ***i*** .

```
A = point( -0.23 , -0.33 )  { (-0.67,-0.37) };
B = point( 4.24 , 1.84 )  { (0.37,-0.43) };
sAB =segment( A , B );
p1 =perpendiculaire( A , sAB )  { i }
perpBsAB =perpendiculaire( B , sAB )  { i }
var x =AB { 4.96888317431593 };
D =pointsur( p1 , x )  { -0.45,-0.07) };
ceBA =cercle( B , A )  { i };
C =intersection( perpBsAB , ceBA , 1 );
```

Et la figure devient :



☞ *Terminer en construisant les segments manquants.*



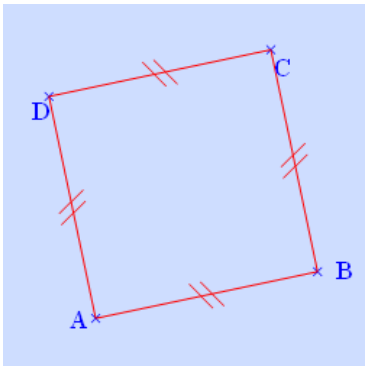
Mise en forme de l'objet

On peut si on le désire modifier l'aspect de l'objet, comme on le constate à l'aide du bouton "mise en forme" : 

A l'aide de cette boîte, on peut changer :

- la couleur des objets,
- le style des traits,
- la mise en forme du texte (noms des points, nom des droites et textes),
- l'aspect des points,
- le codage des segments, milieux, médiatrices et angles.

Si on clique dans une zone vide de la figure alors le curseur reprend sa forme par défaut, ce qui permet de sortir du mode mise en forme.
On peut déplacer cette boîte de dialogue n'importe où sur la scène et continuer à travailler sur la figure. En cliquant à nouveau sur le bouton "mise en forme", on masque la boite de dialogue.



La mise en forme de la figure effectuée, le script a été modifié.

```
A = point( -0.87 , 0.77 )  { (-0.67,-0.37) };
B = point( 4.24 , 1.84 )  { (0.37,-0.43) };
sAB = segment( A , B )  { rouge , \\ };
p1 = perpendiculaire( A , sAB )  { i };
perpBsAB = perpendiculaire( B , sAB )  { i };
var x =AB { 5.2182457673905 };
D = pointsur( p1 , x )  { (-0.45,-0.07) };
ceBA = cercle( B , A )  { i };
C = intersection( perpBsAB , ceBA , 1 );
sBC = segment( B , C )  { rouge , \\ };
sCD = segment( C , D )  { rouge , \\ };
sDA = segment( D , A )  { rouge , \\ };
```

La figure est terminée !

Le carré est dynamique. Il est commandé par un segment et ses deux extrémités : [AB].
On ne peut donc pas déplacer les autres objets.

On aurait également pu construire un carré à partir de son centre et un sommet ... **à vous de jouer !**

2.2 Les fonctions avancées du script

2.2.1 Les boucles

a) La notion de boucle

Les boucles sont utilisées dans la plupart des langages de programmation. Elles permettent dans ce cas d'effectuer des opérations un certain nombre de fois.

Dans un script TracenPoche les boucles permettent de répéter la construction d'un ou plusieurs objets qui seront indicés par une variable dont on fixera la borne supérieure (la borne inférieure ayant obligatoirement pour valeur 1).

La forme générale d'une boucle dans TracenPoche est :

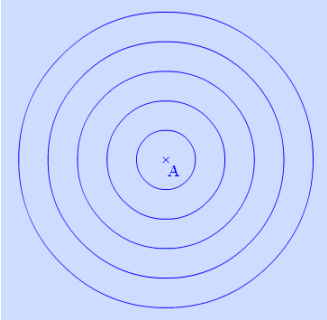
```
for i=1 to n do ;
  script tep (utilisant la variable i )
end ;
```

Le nombre ***n*** est un nombre fixé au départ et non une variable.

b) Quelques exemples de boucles

- Voici un premier script utilisant de manière très simple les boucles dont on voit le résultat à gauche :

```
@options;

@figure;
  A = point( 1.33 , 0.53 );
  for i=1 to 5 do;
    c[i] =cerclerayon( A , [i] );
  end;
```



Ce script (dont le seul intérêt est de vous montrer un premier usage des boucles) montre la construction de 5 cercles de centre A. Ces cercles ont pour noms c1, c2, c3, c4 et c5 et ont pour rayons respectifs 1, 2, 3, 4 et 5.

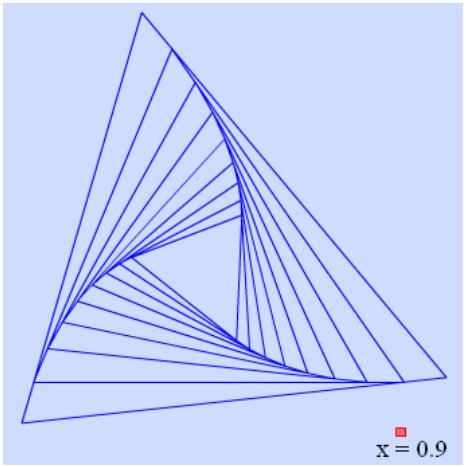
On remarque que la variable **i** (que l'on fait varier de 1 à 5) est écrite entre crochets à l'intérieur du script quand elle est utilisée : **[i]**
Dans ce script elle est utilisée comme indice pour nommer les différents cercles : **c[i]**, mais aussi comme valeur numérique du rayon : **[i]** dans **cerclerayon(A , [i])** :

- quand i vaut 1, le cercle s'appelle c1 et a pour rayon 1,
- quand i vaut 2, le cercle s'appelle c2 et a pour rayon 2,
- ...

- Une boucle peut utiliser une variable, un entier ou un réel défini en dehors de celle-ci.

Comme par exemple dans le script suivant :

```
@options;

@figure;
  A0 = point( -5.83 , -2.25 )  { i };
  B0 = point( 3.84 , -1.21 )  { i };
  C0 = point( -3.1 , 7.11 )  { i };
  p0 =polygone( C0 , A0 , B0  );
  x =reel( 0.9 , 0 , 1 , 0.1 )  { noir , (2.69,-
2.45) };
  for i = 1 to 8 do;
    A[i]=pointsur(B[i-1],C[i-1],x){i};
    B[i]=pointsur(C[i-1],A[i-1],x){i};
    C[i]=pointsur(A[i-1],B[i-1],x){i};
    p[i]=polygone( C[i] , A[i] , B[i]  );
  end;
```

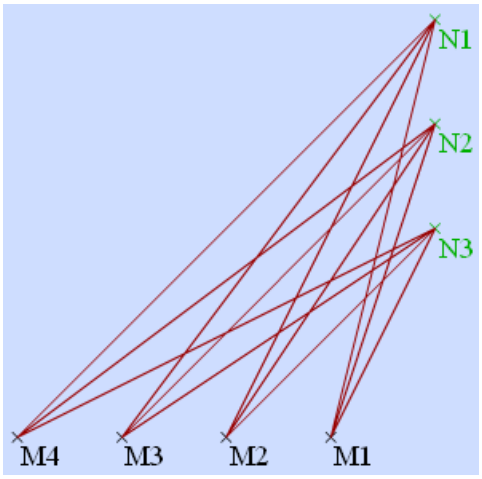


On a défini un réel x compris entre 0 et 1.
Sur les côtés du triangle A0B0C0 (les noms des points sont ici invisibles) on place les points A1, B1 et C1 ayant pour abscisse x. On trace alors le triangle A1B1C1 nommé p1.
Ainsi de suite jusqu'au triangle p8.

On peut ensuite modifier la valeur de x à la souris pour modifier la position des points sur chacun des côtés.

- On peut également créer une boucle à l'intérieur d'une autre boucle :

```
@options;

@figure;
  for i = 1 to 4 do ;
    var x[i] =  - 2*[i] ;
    M[i] = point (x[i], 0) {noir} ;
  end ;
  for j = 1 to 3 do ;
    var y[j] = 10 - 2*[j] ;
    N[j] = point (0, y[j]) {vert};
  end ;
  for i = 1 to 4 do ;
  for j = 1 to 3 do ;


s[i_j]=segment(M[i],N[j]){rougefonce}
;
  end ;
  end ;
```



2.2.2 Les macro-constructions ou macros

a) La notion de macro

Dans une figure TracenPoche, une macro est une sorte de fonction au sens informatique du terme qui, à partir d'objets de départ fournit un objet résultant qui est le dernier objet construit suivant cette macro.
Quand on crée une macro, c'est cette fonction qu'on bâtit.
On peut ensuite utiliser cette macro dans d'autres figures.

Une macro est faite pour être utilisée dans n'importe quelle figure. Elle doit donc être autonome. De plus, il est clair que l'objet résultant construit par une macro doit être déterminé par les objets de départ.

Le logiciel TracenPoche possède un grand nombre de boutons mais aucun qui permette de construire un triangle équilatéral, un carré, le centre de gravité d'un triangle ... la notion de macro va permettre d'y remédier.

La forme générale d'une macro est :

```
fonction nom_fonction(paramètres) ;
  script tep (pouvant utiliser les paramètres)
retourne nom_du_dernier_objet_créé ;
```

La macro doit se situer dans un fichier texte situé dans le même répertoire que le fichier tracenpoche.swf (voir **2.3 TracenPoche et ses fichiers**).

Cette fonction est appelée à partir du script principal à l'aide de la commande :

```
objet = nom_fonction(paramètres) {options};
```

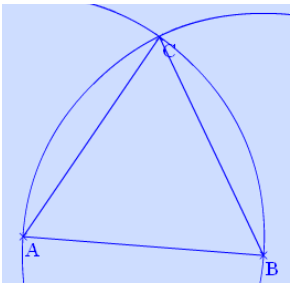
b) Création d'une macro

La fabrication d'une macro se fait en général à partir d'un exemple dans une figure : on part d'un objet (point, droite, cercle ...) dont la construction servira d'exemple d'objet résultant. Cet objet a été construit directement ou indirectement à partir d'autres objets appelés paramètres.

Exemple : Triangle équilatéral.

On commence par créer un script dans lequel on construit un triangle équilatéral :

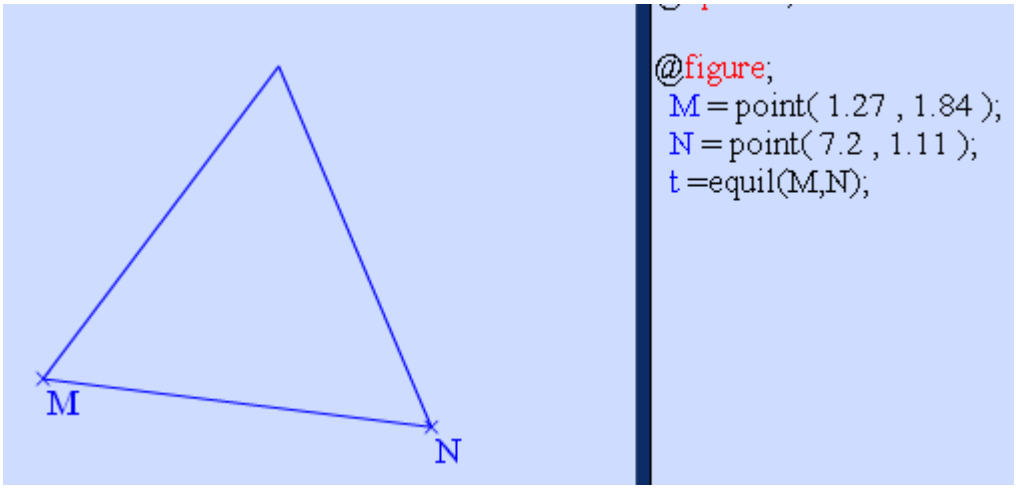
```
@options;

@figure;
  A = point( -1.54 , 1.14 );
  B = point( 5.12 , 0.62 );
  ceAB =cercle( A , B );
  ceBA =cercle( B , A );
  C =intersection( ceBA , ceAB , 2 );
  p =polygone( A , B , C  );
```



Les paramètres sont les points A et B et l'objet résultat est le polygone p.
La macro correspondante que l'on nomme **equil** se présentera donc sous cette forme :

```
fonction equil(A,B);
  ceAB =cercle( A , B );
  ceBA =cercle( B , A );
  C =intersection( ceBA , ceAB , 2);
  p =polygone( A , B , C  );
retourne p;
```

Voici ce que cela donne si on utilise la fonction **equil** dans une figure TracenPoche dans laquelle la macro a été déclarée :



La commande **`t =equil(M,N);`** a construit un triangle équilatéral. Les points M et N jouent ici les rôles des points A et B.

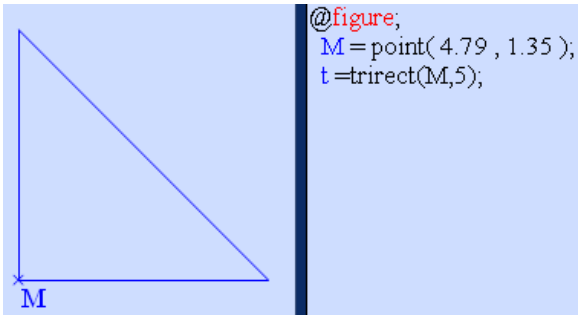
c) Quelques précisions sur les macros

- Les fonctions ne peuvent pas créer l'objet **`var`** par contre elles peuvent utiliser des variables, des entiers ou des réels définis plus tôt dans le script.

Exemple : Construction d'un triangle rectangle et isocèle
Voici la fonction :

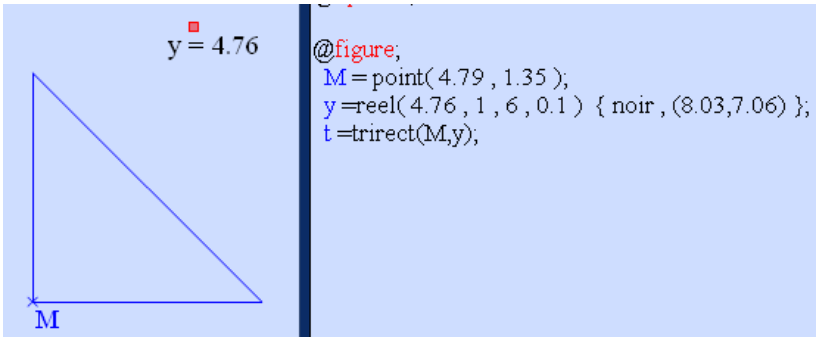
```
fonction trirect(A,x);
  cerAx =cerclerayon( A , x );
  B =pointsur( cerAx , 360 );
  sAB =segment( A , B );
  perpAsAB =perpendiculaire( A , sAB );
  C =intersection( perpAsAB , cerAx , 1 );
  p =polygone( A , B , C  );
retourne p;
```

On peut l'utiliser de cette manière :



Après voir placé le point M, à l'aide de la fonction trirect, on construit un triangle rectangle isocèle en M dont les côtés de l'angle droit mesurent 5 unités.

Ou en définissant une variable avant d'utiliser la fonction :



La longueur des côtés de l'angle droit est définie par la variable y que l'on peut modifier en utilisant la souris.

- Une fonction peut appeler une autre fonction.

Exemple : Centre de gravité d'un triangle.

On crée une première fonction qui construit une médiane (ici la médiane issue de A dans le triangle ABC) :

```
fonction mediane(A,B,C);
  I=milieu(B,C);
  s=segment(A,I);
retourne s;
```

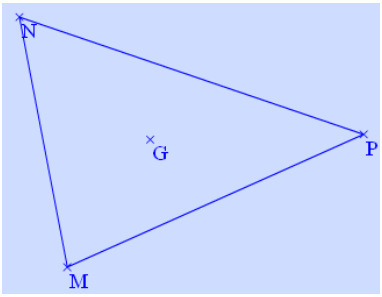
Puis une 2ème fonction qui crée 2 médianes et leur point d'intersection :

```
fonction gravite(A,B,C);
  s=mediane(A,B,C);
  t=mediane(B,A,C);
  E=intersection(s,t);
retourne E;
```

On voit que la fonction gravite fait appel à la fonction mediane en lignes 4 et 5.

Ainsi le script suivant après voir été actualisé donnera la figure de droite :

```
@options;

@figure;
  M = point( -1.97 , -0.51 );
  N = point( -3.23 , 6.06 );
  P = point( 5.83 , 2.98 );
  polyNMP =polygone( N , M , P  );
  G=gravite (M,N,P);
```



2.2.3 Le mode pas à pas

Le mode Pas à Pas (ou Étape par Étape) permet, à l'aide d'une zone de pilotage, de voir la construction d'une figure par blocs ou éventuellement objets par objets.

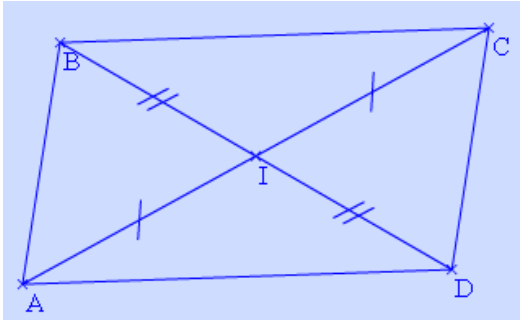
Un bloc est constitué de un ou plusieurs objets. Pour créer un bloc, il faut ajouter la balise **stop** aux options du dernier objet qui le constitue.

Le mode pas à pas permet d'étudier une configuration.

Sans pose de balise, il permet de reprendre la procédure de construction, objet par objet. Une sorte d'historique de la construction qui peut être associé à la lecture en parallèle du script.

Avec des balises **{stop}** bien placées, il permet de mettre en évidence les grandes étapes d'une procédure de construction ou alors les étapes géométriques d'une démonstration.
Cela permet d'instaurer un dialogue de classe : on est parti de là, puis on a construit ceci : qu'a-t-on construit ? est-ce pertinent ? qu'a-t-on voulu souligner ?

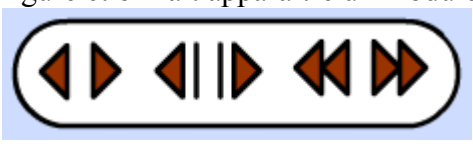
Le script suivant permet de construire un parallélogramme à partir de ses diagonales :




```
@options;

@figure;
  A = point( -4.93 , -1.4 );
  C = point( 4.6 , 3.83 );
  sAC = segment( A , C );
  I = milieu( sAC )  { stop , / };
  B = point( -4.17 , 3.53 );
  D = symetrique( B , I );
  sBI = segment( B , I )  { // };
  sID = segment( I , D )  { stop , // };
  polyABCD = polygone( A , B , C , D  )  { stop };
```

Ce script est constitué de 3 blocs de constructions :
- le 1er bloc comporte les points A et C, le segment [AC] et son milieu I (et son codage),
- le 2ème bloc comporte les points B et D, les segments [BI] et [ID] (et leurs codages),
- le 3ème bloc comporte le polygone ABCD.

En faisant un clic gauche et en appuyant simultanément sur la touche P du clavier dans une zone vide de la figure, on efface la totalité de la figure et on fait apparaître un module de pilotage identique à celui-ci :



Ce module permet de piloter la construction de la figure étape par étape.

Les 2 premiers boutons  permettent d'avancer ou de reculer dans la construction objet par objet (cette fonction n'est possible que si les balises **stop** sont présentes dans le script).

Les 2 boutons suivants  permettent d'avancer ou de reculer dans la construction bloc par bloc.

Les 2 derniers boutons  permettent de revenir au tout début du script (c'est à dire une figure vide) ou d'avancer jusqu'à la dernière balise **stop**.

Il est à noter que, si le dernier objet défini dans le script ne comporte pas l'option **stop**, il ne pourra pas être affiché par le biais du module de pilotage, de même que tous les objets situés après la dernière balise **stop**.

En utilisant à nouveau la combinaison **clic gauche + touche P** du clavier, le module de pilotage disparaît et la figure complète réapparaît.

Si on souhaite qu'à l'ouverture la figure ne soit pas chargée, il suffit d'ajouter dans la section **@config** du script de la figure, la commande **pasapas=oui;**.
Il faut alors faire apparaître le module de pilotage afin de lancer la construction pas à pas.

2.2.4 L'option changement état bloc

Dans la section **@config**, la commande changement état bloc (**chgt_etat_bloc**) permet de modifier les options d'apparence d'un ou plusieurs objets par l'appui simultané sur le bouton gauche de la souris et sur une touche du clavier.

Sa syntaxe est la suivante :
```
chgt_etat_bloc("touche_raccourci",{option_1,option_2,...},objet_1,objet_2,...
{option_i,...},objet_n+1,...);
```

Les paramètres de cette commande sont les suivants :
- **"touche_raccourci"** est la touche du clavier qui, par l'appui simultané avec le bouton gauche de la souris, permettra de changer l'état du ou des objets (elle doit être écrite entre guillemets),
- **{option_1,option_2,...}** est la liste des options caractérisant l'aspect du ou des objets dont le ou les noms vont suivre,
- **objet_1,objet_2,...** est la liste des objets qui prendront l'aspect défini par les options précédentes,
- **{option_i,...}** est une nouvelle liste d'options définissant un état,
- **objet_n,...** est une nouvelle liste d'objets qui prendront l'aspect défini précédemment.

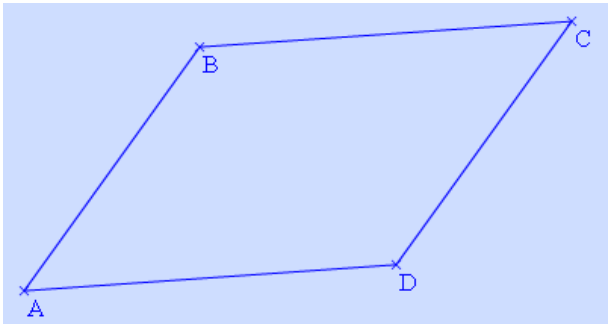
On peut mettre autant de couples liste d'options / liste d'objets que l'on souhaite dans une seule et même commande de changement état bloc.

Le ou les objets dont l'apparence a été modifiée reprendront leur apparence initiale dans deux cas :
- après avoir effectué la même manipulation (souris + clavier),
- ou si une autre commande de changement état bloc est activée par la combinaison clic gauche et autre touche de raccourci.

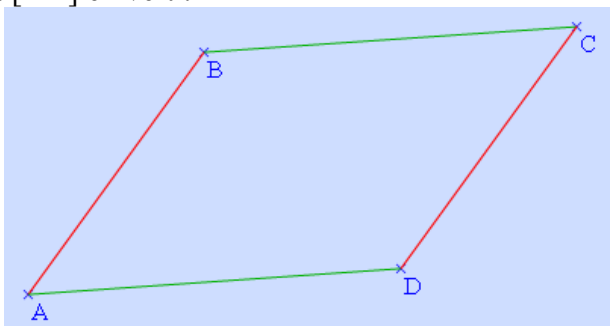
Les options que l'on peut modifier dépendent des objets auxquels elles s'appliquent :
- pour les points et lignes on peut modifier : la couleur, la visibilité (i ou v), le style (croix0, croix1, croix2, croix3, rond0, rond1, rond2, rond3 pour les points; 1, 2, 3, 7, 8, 9 pour les lignes), la présence du nom (avecnom, sansnom), la trace (trace, pastrace),
- pour les cercles on peut modifier : la couleur, la visibilité (i ou v) , le style (1,2,3,7,8,9),
- pour les polygones on peut modifier : la couleur, la visibilité (i ou v) , le style (1, 2, 3, pleinxx).

Exemple :
```
@options;
  chgt_etat_bloc("a",{rouge},sAB,sCD,{vert},sBC,sDA);
  chgt_etat_bloc("b",{v},O,sAC,sBO,sOD);

@figure;
```

```
A = point( -6.1 , -2.73 );
C = point( 5.97 , 3.2 );
sAC = segment( A , C )  { i };
O = milieu( sAC )  { / , i };
B = point( -2.23 , 2.63 );
D = symetrique( B , O );
sBO = segment( B , O )  { // , i };
sOD = segment( O , D )  { // , i };
sAB = segment( A , B );
sBC = segment( B , C );
sCD = segment( C , D );
sDA = segment( D , A );
```

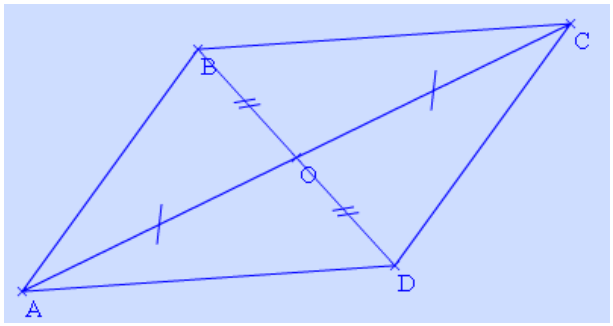


En effectuant la combinaison **clic gauche + touche A**, les segments [AB] et [CD] sont coloriés en rouge et les segments [BC] et [DA] en vert :



En effectuant à nouveau la combinaison **clic gauche + touche A**, le parallélogramme reprend son aspect initial.

En effectuant la combinaison **clic gauche + touche B**, le point O et les segments [AC], [BO] et [OD] deviennent visibles :



Si on utilise la combinaison **clic gauche + touche A** à cet instant, la figure prendra le même aspect qu'après avoir effectué cette manipulation pour la première fois.
C'est à dire que :
- les segments [AB] et [CD] sont coloriés en rouge et les segments [BC] et [DA] en vert,
- le point O et les segments [AC], [BO] et [OD] sont invisibles.

2.2.5 L'objet VARSI et la fonction µ

a) Le principe

Le **VARSI** est un objet **VAR** dont la valeur dépend d'une condition. Sa syntaxe est :

```
varsi  z  =  [ condition , EA1 ou "TEXTE1" , EA2 ou "TEXTE2"  ];
```

ou EA désigne une expression algébrique.

Si la condition est vraie, alors z prendra la valeur EA1 ou "TEXTE1", si elle est fausse, alors z prendra la valeur EA2 ou "TEXTE2".

Un objet VARSI peut-être utilisé en tant qu'opérateur de conditionnement d'existence d'objet. C'est là qu'intervient la fonction µ. Voici la syntaxe :

```
varsi z = [condition,1,0];
nom =  µ(z) typeobjet(paramètres);
```

Cet objet ne sera construit (existera) que si le **VARSI** est différent de 0.

b) Syntaxe des conditions dans un VARSI

- Le premier type de condition est de la forme : **EA opérateur EA**.
  Les opérateurs utilisables sont =, <>, <, >, <= et >=.

  Dans le cas des opérateurs = et <>, il faut préciser la précision souhaitée grâce à une syntaxe de la forme _x%, avec x exprimé en centièmes d'unité.

  Par exemple : `varsi z =[AB=2_1%,3,4];`

  La condition AB=2 est vraie si AB=2 à 0,01 près. Et dans ce cas, z vaut 3, sinon z vaut 4.

  On peut entrer n'importe quelle expression algébrique, aussi bien dans la condition que dans les valeurs du **VARSI**.
  L'exemple suivant en est la preuve, et c'est son seul intérêt :

  ```
  varsi z =[AB+BC=5_1%,ln(5)-AB,2*tan(x)];
  ```

- Le second type de condition est : **point sur (x,y)** ou **point sur point** ou **point appartient d** ou **point appartient AB**.
  Pour chacune de ces conditions, il faut également préciser le degré de précision souhaité à l'aide de la syntaxe _x%.

  1er exemple : `varsi z =[BsurA_30%,1,0] ;`
  z vaut 1 si le point B est en A à 0,3 près, donc si la distance entre A et B est inférieure ou égale à 0,3. Sinon,z vaut 0.

  2ème exemple : `varsi z =[Aappartientd_20%,1,0] ;`
  z vaut 1 si le point A appartient à la droite d à 0,2 près, donc si la distance entre le point A et la droite d est inférieure ou égale à 0,2. Sinon,z vaut 0.

c) Utilisation des VARSI

- ***Comme une autre variable***.
  La valeur prise par un **VARSI** pouvant être aussi bien une expression algébrique qu'un texte, on peut utiliser le **VARSI** de différentes manières.

  On peut tout d'abord l'utiliser comme tout autre objet **VAR**.

```
@options;

@figure;
  A = point( -2 , 0.3 );
  B = point( 0 , 1 );
  cerayA3 = cerclerayon( A , 3 )  { grisfonce };
  varsi z =[AB<3,AB,-AB] { 2.11896201004171 };
```

```
M = point( 3 , z )  { noir , rond2 };
```
Le **VARSI** z sert ici à définir l'ordonnée du point M.

- ***Valeurs texte.***

  Un **VARSI** peut prendre également des valeurs texte, que l'on peut afficher en mettant le nom de la variable entre $ comme un pour un objet **VAR**.
  En voici une illustration :

```
@options;

@figure;
  O = point( 0 , 0 );
  cerayO3 = cerclerayon( O , 3 );
  M = point( 2.68 , 2.23 )  { (-0.7,-0.7) };
  varsi x =[OM<=3,"M est à l'intérieur","M est à l'extérieur"];
  texte1 = texte( -2 , 5 , "$x$")  { rouge , dec2 };
```

  Ce script permet d'afficher si le point M est à l'intérieur ou à l'extérieur du disque de centre A et de rayon 3.

- ***Mises en forme conditionnelles.***

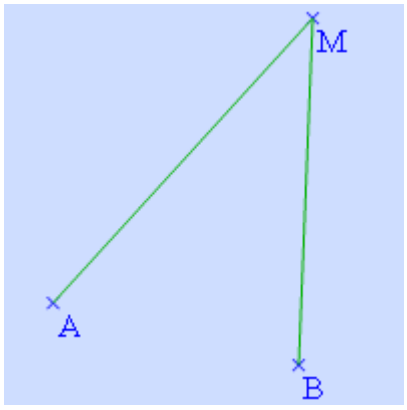
  Cet aspect du **VARSI** permet d'effectuer des mises en forme conditionnelles.
  La syntaxe du **VARSI** est la même, mais les valeurs sont des textes contenant des options de mise en forme séparées par des "-" et non pas des ",".
  Par exemple : `varsi z =[AB<2,"rouge-1","noir-3-v"];`

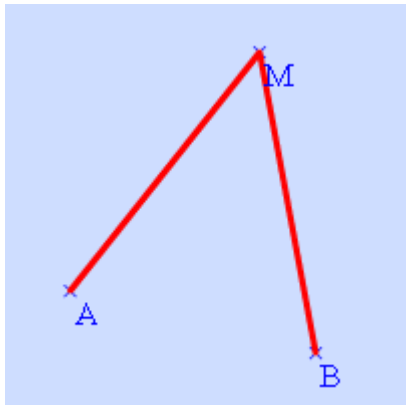
  Ensuite la variable varsi peut être insérée dans les options {} des objets à l'aide des séparateurs $.
  Par exemple : `C = point( 3 , 0.5 )  {bleu,2,$z$};`
  Les mises en forme contenue dans z écraseront les mises en formes bleu et 2.

  Les options de mise en forme que l'on peut conditionner sont la couleur, le style, la visibilité et l'option trace.

  Dans l'exemple suivant, l'aspect des segments [AM] et [BM] changera selon qu'ils aient ou pas la même longueur à 0,05 près.

```
@options;

@figure;
  A = point( -3 , -1 );
  B = point( 1 , -2 );
  M = point( 4 , 3 );
  varsi z =[AM=BM_5%,"rouge-3","vert"] { vert };
  sAM = segment( A , M )  {$z$ , vert };
  sBM = segment( B , M )  {$z$ , vert };
```





d) La fonction µ

Cette fonction utilise pour variable un **VARSI**, et permet de conditionner l'existence d'un objet.

La syntaxe complète est :

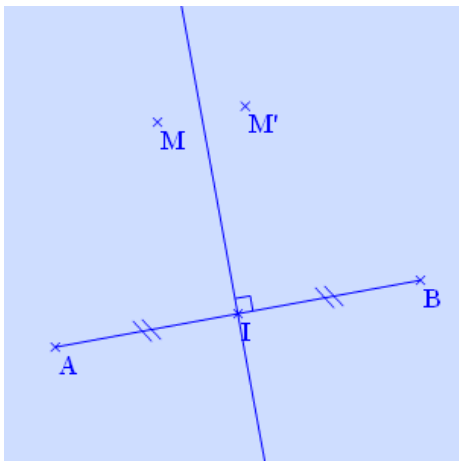
```
varsi z = [condition,1,0] ;
nom = µ(z) typeobjet(paramètres) ;
```

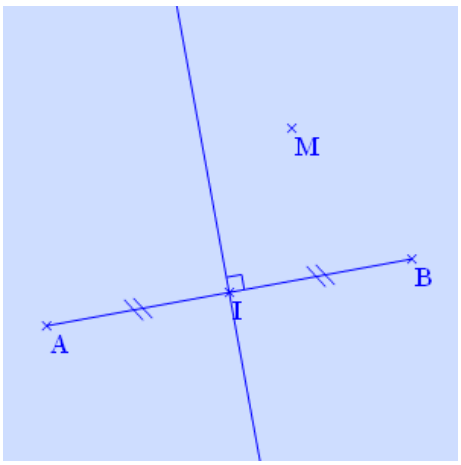
L'objet ne sera construit que si la valeur de z est différente de 0.

On peut conditionner l'existence des points, des lignes, des cercles, des textes, des polygones des angles et des transformations.

Dans l'exemple suivant, le symétrique de M par rapport à la médiatrice de [AB] n'existe que si A appartient au demi-plan contenant A :

```
@options;

@figure;
  A = point( -4.7 , -1.6 );
  B = point( 4.8 , 0.1 );
  sAB = segment( A , B );
  I = milieu( sAB )  { \\ };
  medsAB = mediatrice( sAB );
  M = point( -2.5 , 3.3 );
  varsi z =[MA<MB,1,0] { 1 };
  M' = µ(z) symetrique( M , medsAB );
```





e) Imbrication des VARSI

Il est possible d'imbriquer les **VARSI** les uns dans les autres.
Seuls 2 niveaux sont autorisés, dans les autres cas il suffit d'utiliser des **VARSI** intermédiaires !
Il y a donc 3 types d'imbrications différentes :

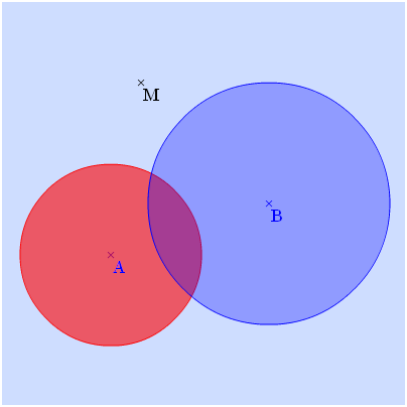
- `varsi z = [ condition1 , [ condition2 , EA1 ou "TEXTE1" , EA2 ou "TEXTE2" ] , EA3 ou "TEXTE3" ];`
- `varsi z = [ condition1 , EA1 ou "TEXTE1" , [ condition2 , EA2 ou "TEXTE2" , EA3 ou "TEXTE3" ] ];`
- `varsi z = [ condition1 , [ condition2 , EA1 ou "TEXTE1" , EA2 ou "TEXTE2" ] , [ condition3 , EA3 ou "TEXTE3" , EA4 ou "TEXTE4" ] ];`

Ces imbrications permettent de programmer 3 types de booléens :

- le booléen (condition1 ET condition 2) :
  `varsi z = [ condition1, [condition2, 1, 0], 0 ] ;`
- le booléen (condition1 OU condition 2) :
  `varsi z = [ condition1, 0, [condition2, 1, 0] ] ;`
- le booléen (condition1 XOR condition 2), qui correspond au "ou exclusif" :
  `varsi z = [ condition1, [condition2, 0, 1], [condition2, 1, 0] ] ;`

Dans l'exemple ci-dessous, le VARSI z ne prendra pour valeur 1 que si le point M appartient à l'intersection des 2 disques :

```
@options;

@figure;

  A = point( -2.5 , -1 );
  B = point( 2.5 , 1 );
  c1 = cerclerayon( A , 3 )  { rouge , plein60 };
  c2 = cerclerayon( B , 4 )  { plein30 };
  M = point( -1.5 , 5 )  { noir };
  varsi z =[AM<=3,[BM<=4,1,0],0] { 0 };
  t = µ(z) texte( -5 , 5 , "Dans l'intersection !")
{ dec2 };
```



2.2.6 L'objet PointAimante

a) Principe

Un **PointAimante** est un point libre qui est "attiré" par certaines positions privilégiées dès qu'il en est proche.

Ceci permet de forcer un point à appartenir à un cercle, à une droite ou à être confondu avec un autre point.
Il y a donc 3 types de **PointAimante** selon que l'on veuille aimanter le point sur un cercle, une droite ou un autre point.

b) Les syntaxes

La syntaxe générale d'un **PointAimante** est :

`nom = pointaimante(abscisse,ordonnee, condition_x% );`

Si la condition est vraie à un certain degré de tolérance défini à l'aide du x%, alors le point prendra les coordonnées proches telles que la condition soit vraie rigoureusement.
C'est à dire que le point s'aimantera sur les coordonnées voisines pour lesquelles la condition est vraie.
La condition doit toujours commencer par le nom du point.

- ***Point aimanté sur un cercle.***

La syntaxe est :

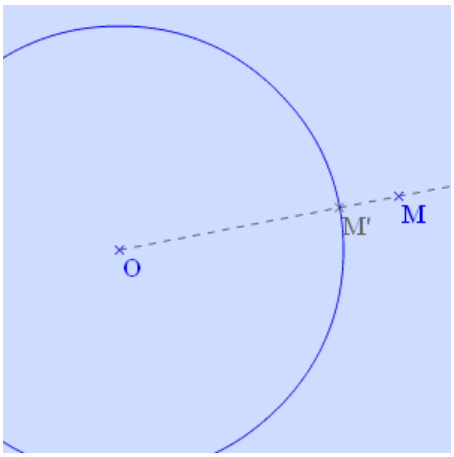
```
M = pointaimante(a,b, MO = expression algébrique_x%  );
```

M est aimanté sur le cercle de centre O et de rayon la valeur de l'expression algébrique.

Par exemple, dans le script suivant, le point M est aimanté sur le cercle de centre O et de rayon 5 :

```
@options;

@figure;
  O = point( 0 , 0 );
  cerayO5 = cerclerayon( O , 5 );
  M = pointaimante( 6.2 , 1.2 , MO=5_10% );
```

En indiquant `MO=5_10%`, on précise que si la distance MO est comprise entre 4,9 et 5,1 (c'est à dire 5 – 0,1 et 5 + 0,1) alors le point M se positionnera sur le cercle, c'est à dire tel que la condition MO = 5 soit rigoureusement vraie.
En fait le point M s'aimante sur le point M' qui est l'intersection du cercle et de la demi-droite [OM), comme le montre le dessin ci-dessous :



Si on écarte le point M de plus de 0,1 du cercle, alors M redevient un point libre.

L'expression algébrique qui définit le rayon du cercle peut-être aussi compliquée qu'on le souhaite.

Dans l'exemple suivant, le point M est aimanté sur le cercle circonscrit au triangle ABC.
C'est pour cela que l'on a `MO = p/(4*a)` où p = AB*AC*BC et a est l'aire du triangle ABC.
Le point M s'aimantera sur le cercle si il est à moins de 0,2 unités de celui-ci.

```
@options;

@figure;
  A = point( 0 , 7 );
  B = point( 5 , 1.5 );
  C = point( -4 , -3 );
  var p =AB*AC*BC { 805.550277760488 };
  var a =aire(ABC) { 36 };
  t = polygone( A , B , C  );
  m1 = mediatrice( A , B )  { i };
```

```
m2 = mediatrice( B , C )  { i };
O = intersection( m1 , m2 )  { i };
c = cercle( O , A )  { grisfonce , 7 };
M = pointaimante( 6 , -2 , MO=p/(4*a)_20% );
```

- ***Point aimanté sur une droite***

La syntaxe est :

```
M = pointaimante(a,b, M appartient d_x% );
```

ou

```
M = pointaimante(a,b, M appartient AB_x% );
```

M est aimanté sur la droite d ou sur la droite (AB) (la droite (AB) n'a pas besoin d'être définie dans le script).

Par exemple, dans l'exemple suivant, le point C est aimanté sur la droite (AB) (qui n'est définie dans le script) et D est aimanté sur la médiatrice de [AB].

```
@options;

@figure;
  A = point( -5.3 , -1.6 );
  B = point( 2.2 , -2.8 );
  d = mediatrice( A , B )  { vertfonce , 7 };
  C = pointaimante( 4 , -2.6 , C appartient AB_40% )  { rouge };
  D = pointaimante( 0.8 , 3.8 , D appartient d_20% )  { vertfonce };
```

Le paramètre de x% correspond à la distance (en centième d'unité) entre le point et la droite pour que celui-ci soit aimanté.
Ainsi, dans l'exemple précédent, `C appartient AB_40%`, indique que le point C sera aimanté sur la droite (AB) si la distance entre C et (AB) est inférieure ou égale à 0,4.

- ***Point aimanté sur un autre point***

La syntaxe est :

```
M = pointaimante(a,b, M sur A_x% );
```

ou

```
M = pointaimante(a,b, M sur (EA1,EA2)_x% );
```

ou EA désigne une expression algébrique.

M est aimanté sur le point A ou sur le point de coordonnées (EA1,EA2).

Dans l'exemple suivant, le point M est aimanté sur le point I, milieu de [AB]. Le point N est aimanté sur le point de coordonnées (2*x,1+y), où x et y sont les coordonnées de A, donc ici sur le point de coordonnées (2;6) (on pourra faire apparaître le repère dans TeP).

```
@options;

@figure;
  A = point( 1 , 5 );
  B = point( -3.5 , -0.5 );
  sBA = segment( B , A );
  I = milieu( sBA );
  var x =abscisse(A) { 1 };
  var y =ordonnee(A) { 5 };
  M = pointaimante( 0.87 , 2.03 , M sur I_20% );
  N = pointaimante( 2.7 , 0.23 , N sur (2*x,1+y)_30% );
```

Dans cet exemple, **`M sur I_20%`**, indique que M sera aimanté sur I si il se trouve à une distance inférieure ou égale à 0,2 du point I.

2.3 TracenPoche et ses fichiers

2.3.1 Les différents fichiers

Dans ce paragraphe, on suppose que tracenpoche.swf est dans un répertoire …\TEP
Dans ce répertoire, on trouve les fichiers suivants :

tracenpoche.swf ou ***tracenpoche.exe***	le programme
base.tep	ce fichier contient les fichiers figures devant être chargés au lancement de TeP
figure1.txt ***figure2.txt*** ***figure3.txt*** ***figure4.txt***	figures
base.txt	figure de base, chargée automatiquement au lancement de TeP
macro.tep	ce fichier contient les fichiers macros devant être chargés au lancement de TeP
macros1.txt	fichier regroupant une première liste de macros
macros2.txt	fichier regroupant une deuxième liste de macros

2.3.2 Structure du fichier base.tep

C'est un simple fichier texte qui sert de bibliothèque de scripts au logiciel.

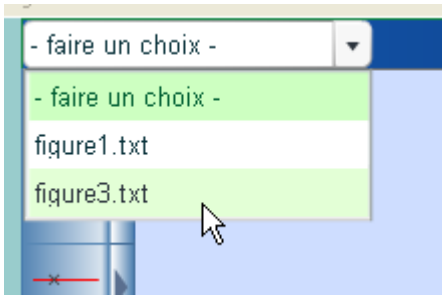
Si on désire que seules les figures **`figure1.txt`** et **`figure3.txt`** soient chargées au lancement de TracenPoche, le contenu de **`base.tep`** devra être le suivant :

```
figure1.txt ;
figure3.txt ;
```

Au lancement de TeP, le fichier **`base.txt`** sera chargé automatiquement. Ce fichier est un fichier figure comme les autres (voir **2.3.3 Structure d'un fichier figure**).
Si on veut une figure vide au lancement de TeP, soit on supprime **`base.txt`**, soit on ne définit aucune figure dans ce fichier (juste **`@options;`** et **`@figure;`**).

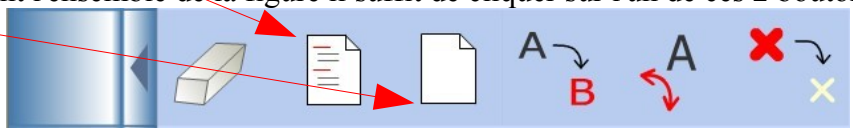
Les fichiers **`figure1.txt`** et **`figure3.txt`** ont été chargés en mémoire car TeP a ouvert puis lu le contenu de **`base.tep`** et connaît donc les fichiers figures qu'il doit charger.
Pour accéder aux figures chargées, cliquer sur le bouton représentant un dossier jaune.

Ceci permet d'afficher une liste déroulante contenant les fichiers figures chargés.



Il suffit alors de cliquer dans cette liste pour charger un fichier donné.

Pour recharger la figure de base (c'est à dire supprimer toute modification depuis son chargement) ou pour effacer complètement l'ensemble de la figure il suffit de cliquer sur l'un de ces 2 boutons.



2.3.3 Structure d'un fichier figure

Ce fichier comporte 5 sections :

```
@ options;
@ figure;
@ analyse;
@ enonce;
@ config;
```

Les sections **@options** et **@figure** sont la réplique exacte des sections analogues dans l'interface de TracenPoche. Quand un fichier est chargé, les deux zones sont remplies et la figure est affichée.

a) La section @enonce

Dans TeP, figure une **zone énoncé.**
Elle est en lecture seule dans l'interface graphique. Elle permet de proposer un texte qui décrit l'activité chargée. Ce texte est au format HTML pour des mises en forme simples.

Par exemple :
Aire d'un parallélogramme
permet d'obtenir :

Il est à noter que pour que les accents soient conservés dans l'énoncé, comme dans les commentaires de la zone script, il faut enregistrer le fichier au format UTF-8 (Bloc-notes ou Notepad sous MS Windows XP, enregistrer, choisir UTF-8 dans la liste Codage).

b) La section @config

Chaque figure TeP se lance en configurant l'interface de TracenPoche grâce au contenu (présent ou absent) de cette section.
La section **@config** permet de régler les couleurs du fond des zones, de positionner et de régler les tailles de chaque zone, de préciser les boutons disponibles et de préciser les commandes disponibles.

Exemple de section **`@config;`** :

```
@config;
couleurfonddessin=0xceddff;
couleurfondtexte=0xceddff;
couleurfondanalyse=0xceddff;
couleurfondenonce=0xceddff;
couleurtextinfo=0xffffff;
figure=0,0,670,560,ouvert;
script=678,30,344,420,ouvert;
analyse=678,458,344,234,ouvert;
enonce=678,3,340,100,ferme;
info=50,652,620,620;
listetransfos=visible;
actualisescript=oui;
pasapas=non;
boutons=point,pointsur,inter,inter2,milieu,projeteortho,segment,droite,demidroite,polygone,vecteur,droiteparallele,droiteperpendiculaire,mediatrice,bissectrice,droitev,tangente,cercle,cercledia,cerclerayon,arc,angle,symetrique,deftransfo,imagetransfo,texte,mesure,mesureangle,nombres,animation,lieu,supprimer,effacerbase,effacertout,renomme,posnom,invisible,repere,zoomp,zoomm,zoomb,zoombm,deplacement,rapporteur,guide,miseenforme,tepmep,aide,fichier,fiche,ootep,exporter,oooexport,configuration,escape;
commandes=point,milieu,intersection,inter,intercercles,interlignes,projete,symetrique,pointsur,barycentre,image,segment,segmentlong,vecteur,vecteurcoord,droite,droitev,droiteeqr,droiteeq,demidroite,parallele,perpendiculaire,bissectrice,mediatrice,tangente,cercle,cercledia,cerclerayon,angle,arc,entier,reel,calcul,distance,perimetre,mesureangle,mesureangleg,mesurearc,aire,symetrie,reflexion,translation,rotation,homothetie,similitude,polygone,lieu,groupe,fonction;
commandesanalyse=distance,calc,abscisse,ordonnee,coord,ec,er,angle,position,nature,aire,alignés,sto,pgcd,ppcm,simplifie,anglev,exact,dim;
```

Les couleurs :

```
couleurfonddessin=0xceddff;
couleurfondtexte=0xceddff;
couleurfondanalyse=0xceddff;
couleurfondenonce=0xceddff;
couleurtextinfo=0xffffff;
```

Les couleurs sont codées à l'aide d'une représentation hexadécimale : 0x + Rouge + Vert + Bleu.
Chaque couleur a une teinte dont la force est comprise entre 00 et FF. Cette représentation est accessible dans la plupart des logiciels de "dessin" (retouche photo par exemple).
Par exemple 0xff00ff donne du violet (rouge à fond + bleu à fond).

Les zones :

```
figure=0,0,670,560,ouvert;
script=678,30,344,420,ouvert;
analyse=678,458,344,234,ouvert;
enonce=678,3,340,100,ferme;
info=50,652,620,620;
listetransfos=visible;
```

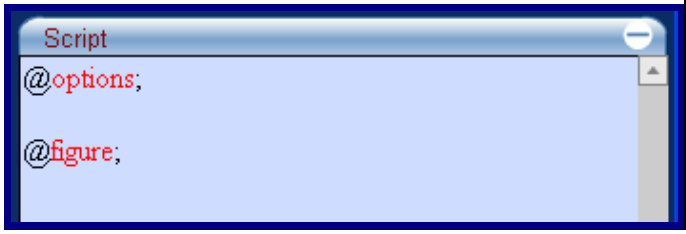
Qui dit positionner dit repérage.
L'écran est repéré comme suit :

- l'origine est en haut à gauche,
- l'axe des abscisse est orienté vers la droite,
- l'axe des ordonnées est orienté vers le bas,

- l'unité est le pixel.

Décodage de la ligne : **`script=678,30,344,420,ouvert;`**
(678,30) donne les coordonnées dans ce repère du coin supérieur gauche de la zone script.
La zone script a une largeur de 344 pixels et une hauteur de 420 pixels.
La zone s'affichera à l'emplacement voulu et sera ouverte (mot clé ouvert)

`script=678,30,344,420,ferme;`
affiche également la zone, mais elle sera fermée (mot clé ferme) : il faudra cliquer sur le bouton + pour l'ouvrir.

Zone script ouverte	Zone script fermée
	

La ligne **`listetransfos=visible;`** indique que la liste des transformations est visible, on pourrait avoir aussi **`listetransfos=invisible;`**

L'actualisation du script :
Comme évoqué dans un chapitre précédent (**2.1.3.3 Les boutons de la zone script**), quand certains scripts sont assez lourds (c'est à dire qu'ils contiennent beaucoup de lignes) il est intéressant de désactiver la réactualisation du script après avoir lâché la souris pour ne pas que la figure devienne, pendant quelques secondes, "inaccessible".
Pour le faire, il suffit d'appuyer sur le bouton 

en bas de la zone de script ou d'appuyer simultanément sur le bouton gauche de la souris et la touche S (comme Script) du clavier.
Par contre si on veut activer cela au chargement d'une figure, il suffit de mettre dans la section **`@config`** : **`actualisescript=non;`**

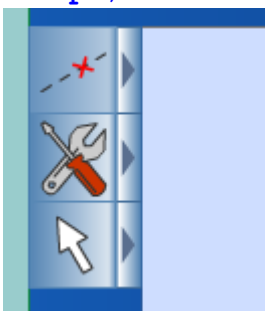
Inversement, il peut parfois être utile d'avoir une actualisation continue du script. C'est à dire, que le script n'est pas actualisé seulement après avoir relâché un objet avec la souris, mais il est actualisé de manière continue pendant qu'un objet est déplacé.
Pour cela il suffit de mettre dans la section **`@config`** : **`actualisationcontinue=oui;`**

Le mode Pas à Pas
Le mode Pas à Pas (ou Étape par Étape) permet, à l'aide d'une zone de pilotage, de voir la construction de la figure par étapes ou par blocs (voir **4.1 Les Options**).
Quand, dans la section **`@config`**, la ligne **`pasapas=oui;`** apparaît la figure n'est pas dessinée au démarrage.
La combinaison clic gauche + touche P du clavier permet de faire apparaître la zone de pilotage là où on a cliqué. On peut ensuite la déplacer.
La ligne **`pasapas=non;`** donne une figure dessinée complètement au démarrage.

Les boutons :
`boutons=..., configuration, escape;`
Ici, on obtient tous les boutons. On aurait pu aussi écrire : **`boutons = tous;`**

Cette ligne permet de spécifier les boutons que l'on désire rendre disponibles au chargement de la figure.
On peut très bien ne rendre disponibles que quelques boutons.

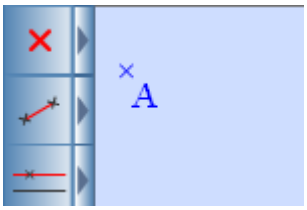
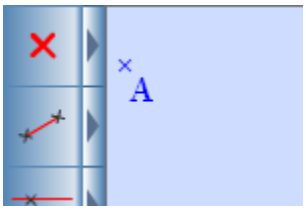
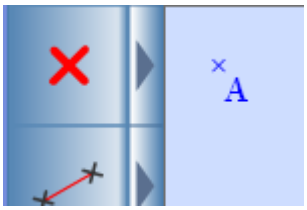
Par exemple :
boutons=pointsur,configuration,escape; donne l'interface visible ci-dessous :



Il est possible d'ajouter une commande supplémentaire avant cette ligne afin de modifier la taille des boutons :
reducbouton=x; avec x qui est un coefficient de réduction ou d'agrandissement de la taille des boutons. Suivant la valeur x choisie les boutons apparaîtront plus grands ou plus petits.

Par exemple, la commande **reducbouton=0.8;** permet d'avoir des boutons ayant des dimensions réduites de 20%.

Par contre, la commande **reducbouton=1.5;** permettra d'avoir des boutons ayant des dimensions augmentées de 50%.

reducbouton=0.8 ;	taille par défaut	reducbouton=1.5 ;
		

Les commandes :
comandes= ... groupe,fonction; ou **commandes=toutes;**
commandesanalyse= ...,exact,dim; ou **commandesanalyse=toutes;**

Ces 2 lignes indiquent qu'on autorise l'ensemble des commandes (mots-clés) du langage reconnu par la **zone Script** et par la **zone Analyse**.
Sinon, il faut lister les commandes autorisées comme on le fait pour les boutons.

c) Remarques particulières

- Si dans la section **@config** on omet une ligne, **analyse= 678,458,344,234,ouvert;** par exemple, TeP n'en conclut pas pour autant que la fenêtre analyse est absente.
  Il en conclut qu'il doit prendre la configuration par défaut pour la fenêtre analyse (une fenêtre analyse s'affiche donc à l'écran).
  Par défaut toutes les zones sont visibles et ouvertes sauf la **zone énoncé**.
  Donc, si on ne veut pas d'une fenêtre analyse, il suffit de l'afficher dans une zone invisible par exemple :

  analyse=2000,370,220,120,ouvert;

`2000` représente une position non affichée à l'écran.

- Si la section `@config` est absente alors TeP chargera la figure dans sa configuration par défaut où toutes les zones sont visibles et ouvertes sauf la `zone énoncé`.
  Un fichier figure ne peut très bien comprendre que les sections `@options` et `@figure` .

- Le fichier `base.txt`

  `base.txt` est un fichier particulier.
  La figure de ce fichier est automatiquement affichée par TeP (sauf si ce fichier est absent, auquel cas c'est la configuration par défaut qui est utilisée) quand on le lance.
  L'intérêt est de pouvoir lancer TeP dans une configuration particulière, sans pour autant afficher une figure.

  Par exemple :

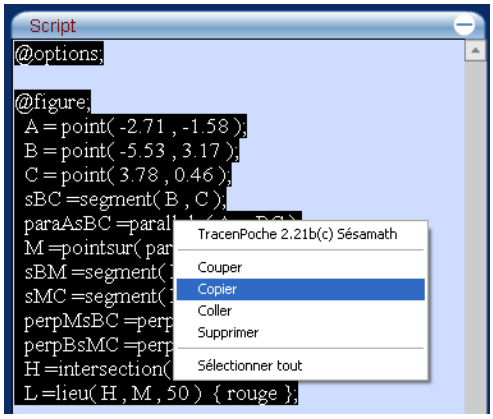
```
@options;
@figure;
@analyse;
@enonce;
@config;
couleurfonddessin=0xceddff;
couleurfondtexte=0xceddff;
couleurfondanalyse=0xceddff;
couleurfondenonce=0xceddff;
couleurtextinfo=0xffffff;
figure=0,0,670,560,ouvert;
script=678,30,344,420,ouvert;
analyse=2000,458,344,234,ouvert;
enonce=2000,3,340,100,ferme;
info=50,652,620,620;
listetransfos=visible;
actualisescript=oui;
pasapas=non;
boutons=point,pointsur,milieu,inter,segment,droite,demidroite,droiteparalle
le,droiteperpendiculaire,cercle,cercledia,arc;
commandes=toutes;
commandesanalyse=toutes;
```

  Ce fichier permet de lancer TeP avec 12 boutons.
  Seuls la figure et le script sont visibles (voir les **2000**).

2.3.4 Sauvegarde et chargement

Pour sauvegarder les figures, il faut copier-coller des fichiers textes manuellement via un éditeur de textes style Bloc-Notes et enregistrer ces fichiers textes dans le répertoire de TeP. Pourquoi ces pirouettes ? Tout simplement parce que Flash, pour des raisons de sécurité, ne sait pas accéder simplement aux disques locaux en écriture (voir les paragraphes **2.5.2** et suivant sur le fonctionnement en ligne)
Deux méthodes sont possibles :

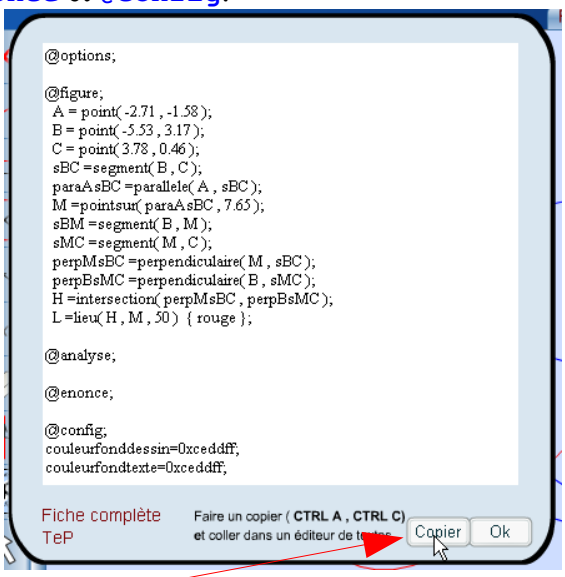
- **Copier / Coller ce qui se situe dans la zone script.**
  On sélectionne le texte dans la zone script, clic droit, on le copie puis on le colle dans un fichier texte.



Mais avec cette méthode seules les section **`@options`** et **`@figure`** seront utilisées.

- **Copier / Coller la fiche complète de la figure.**
  On clique sur le bouton qui permet d'afficher la fiche complète de la figure :

Une fenêtre affiche la fiche complète de la figure, c'est à dire avec les 5 sections : **`@options`**, **`@figure`**, **`@analyse`**, **`@énoncé`** et **`@config`**.



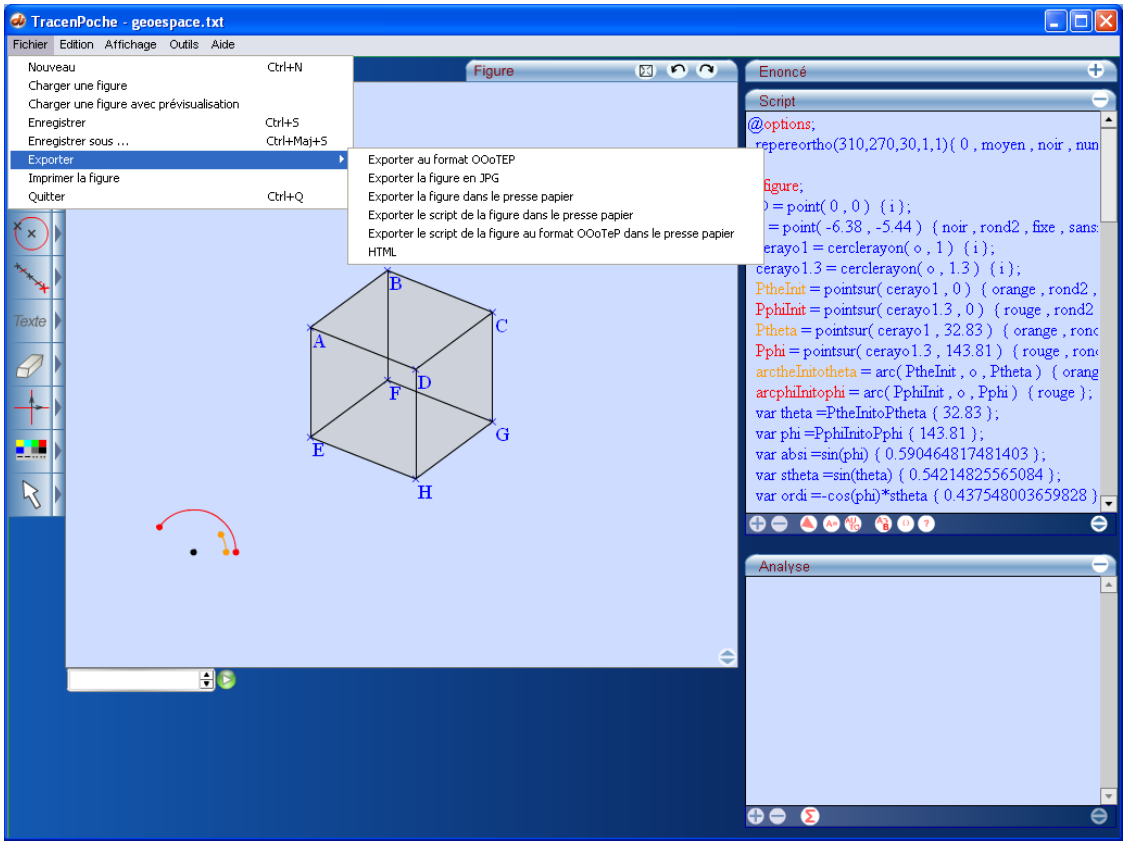
En cliquant sur la bouton "Copier", on copie l'intégralité du texte de la figure.
Il reste à le coller dans un fichier texte (en modifiant éventuellement les différentes sections comme la section **`@config`**).

On rappelle que pour que les accents soient conservés dans l'énoncé, comme dans les commentaires de la zone script, il faut enregistrer le fichier au format UTF-8 (Bloc-notes ou Notepad sous MS Windows XP)

Ensuite on peut construire le fichier **`base.tep`** pour indiquer quels sont les fichiers pourront être chargés dans une session de TeP.

2.4 Le CD-ROM TracenPoche

2.4.1 Présentation du CD



En dehors de l'interface classique de TracenPoche avec ses différentes zones, la version CD du logiciel donne accès à différents menus qui permettent :

- une gestion conviviale des fichiers scripts (sauvegarde et chargement),
- une configuration aisée de l'interface,
- l'exportation sous différentes formes (jpg, presse papier, html),
- l'impression des figures,
- la création de séances élèves comprenant le logiciel TracenPoche version élève (avec des menus restreints) et une bibliothèque de scripts.

2.4.2 Installation du logiciel

L'installation du logiciel TracenPoche s'effectue en exécutant le fichier **setup.exe**. Le répertoire d'installation par défaut est **C:\Program Files\Sésamath\TracenPoche**.

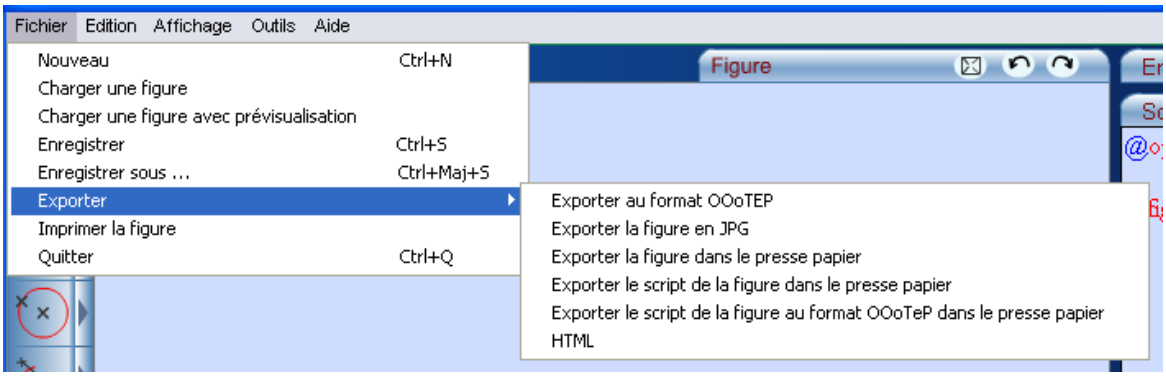
Une fois l'installation effectuée, à l'intérieur de ce répertoire, se trouve, entre autres :

- le fichier **TracenPoche.exe**, qui permet de lancer le logiciel,
- le fichier **ClientUpdate.exe**, qui permet d'effectuer une mise à jour du logiciel (connexion internet requise),
- un dossier **commun** dans lequel on retrouve les fichiers qui permettent de configurer TracenPoche (base.txt, base.tep, macro.tep, ... on peut modifier ces fichiers comme on le fait avec les versions téléchargeables sur le site),
- un dossier **scripts** où se trouvent différents scripts et dans lequel vous pourrez enregistrer vos propres créations.
- un dossier **aide** dans lequel on trouve l'ensemble de l'aide sur TracenPoche au format html.

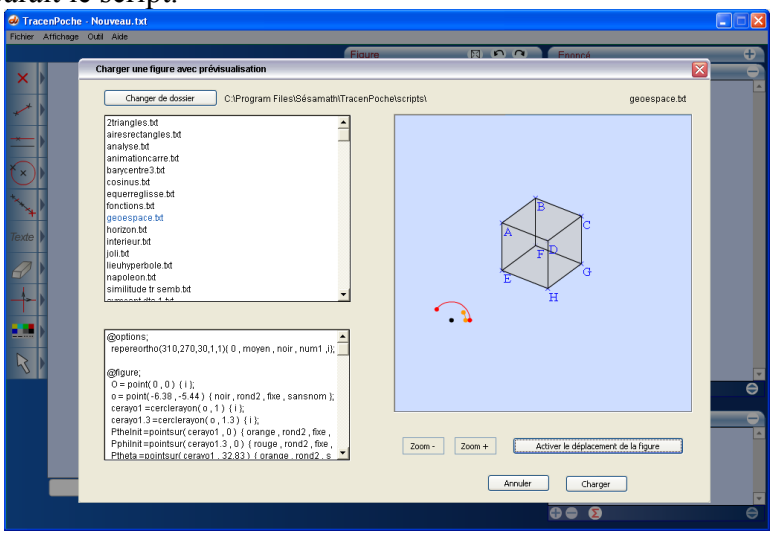
2.4.3 Les menus de l'interface

L'interface du CD-ROM de TracenPoche comporte 4 menus supplémentaires non disponibles avec les versions proposées sur le site : le menu **Fichier**, le menu **Affichage**, le menu **Outil**s et le menu **Aide**.

a) Le menu Fichier.



- **Nouveau** : Efface la figure en cours et charge une figure vide.
- **Charger une figure** : ouvre un explorateur afin de charger un fichier texte contenant le script d'une figure.
- **Charger une figure avec prévisualisation** : explore le contenu d'un répertoire comme précédemment, mais avec un aperçu de la figure dans une fenêtre TepWeb ainsi qu'une fenêtre dans laquelle apparaît le script.



- **Enregistrer** : enregistre le script complet (figure et configuration de l'interface) de la figure en cours dans un fichier texte.

- **Enregistrer sous ...** : enregistre le script complet (figure et configuration de l'interface) de la figure en cours dans un fichier texte sous le nom et dans le répertoire de votre choix.

- **Exporter au format OOoTeP** : enregistre le script de la figure au format OOoTeP (voir 4.3 OOoTeP).

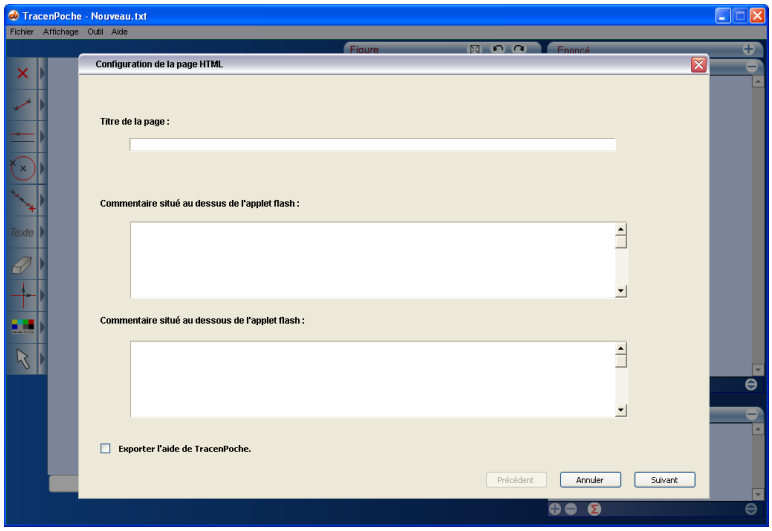
- **Exporter la figure en JPG** : capture l'image de la zone figure affichée et crée un fichier image au format bitmap compressé JPG.

- **Exporter la figure dans le presse papier** : capture l'image de la zone figure affichée et copie l'image non compressée dans le presse-papier.

- **Exporter le script de la figure dans le presse papier** : copie le script complet de la figure dans le presse-papier.

- **Exporter le script de la figure au format OOoTeP dans le presse papier** : copie le script de la figure au format OOoTeP dans le presse-papier.

- **HTML** : permet de générer une page au format html contenant la figure en cours dans une applet TracenPoche (comme sur le site).
  Un fenêtre de configuration permet de donner un titre à la page, et d'ajouter éventuellement du texte au dessus ou en dessous de l'interface TracenPoche (on peut utiliser les codages HTML de mise en forme) .



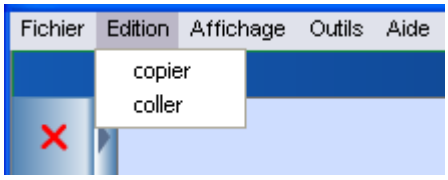
  5 fichiers sont générés :
  - un fichier index.html (la page web en elle-même),
  - un fichier tracenpoche.swf (l'applet insérée dans la page web) ,
  - un fichier base.txt (il contient le script complet de la figure),
  - un fichier base.tep (qui ne référence comme figure que le base.txt)
  - un fichier macro.tep (qui ne contient aucune référence à un fichier macro)

  Ces fichiers sont minimaux, ils peuvent bien sûr être personnalisés avec un bloc-notes (hormis le ficher swf)

- **Imprimer la figure.**

- **Quitter.**

b) Le Menu Edition

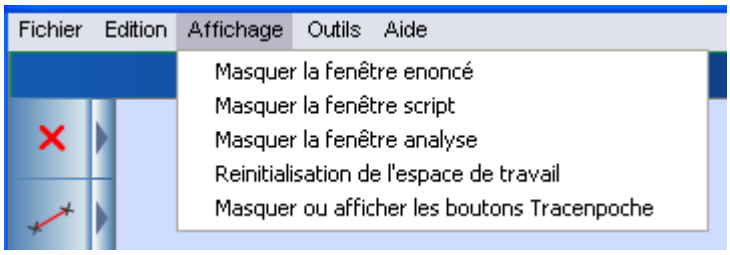


Ce menu permet d'intervenir dans le contenu de la zone script.

- **copier** : copie le texte sélectionné dans la zone script dans le presse-papier.

- **coller** : colle dans la zone script (à l'endroit où se situe le curseur) le contenu du presse-papier.

c) Le menu Affichage



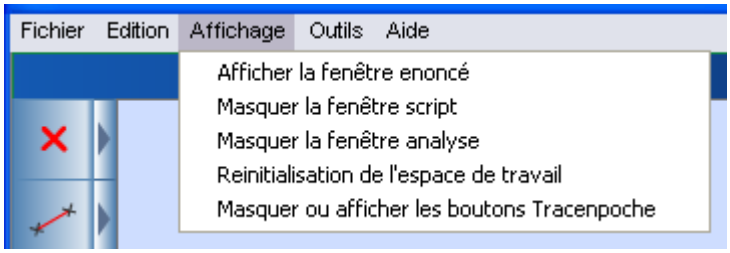
Ce menu permet de configurer l'interface de TracenPoche.

- **Masquer / Afficher une zone.**

Quand on lance le logiciel TracenPoche, l'aspect de l'interface est défini par le fichier base.txt et plus particulièrement par le contenu de la section `@config;`.
Si cette zone est vide ou le fichier absent et dans ce cas l'interface se lance avec sa configuration par défaut.

Ce menu permet de modifier facilement l'interface de TracenPoche.
Par exemple, en cliquant sur Masquer la fenêtre script, non seulement cette zone disparaît de l'interface actuelle, mais le contenu de la section @config; du script de la figure est modifié :
`script=684,25,278,430,ouvert;` devient `script=-3000,-3000,278,430,ouvert;`
Les coordonnées du coin supérieur gauche de la zone script étant (-3000;-3000), cette zone est bien invisible à l'écran.

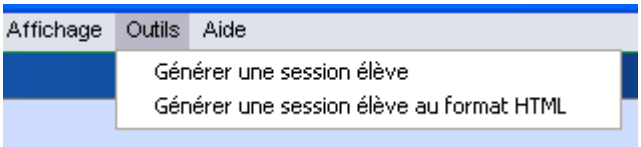
Le menu Affichage a également été modifié et donne maintenant la possibilité d'afficher à nouveau la zone masquée :



- **Réinitialisation de l'espace de travail** : l'interface TracenPoche reprend son aspect initial tel qu'il est défini dans le fichier base.txt ou son aspect par défaut.

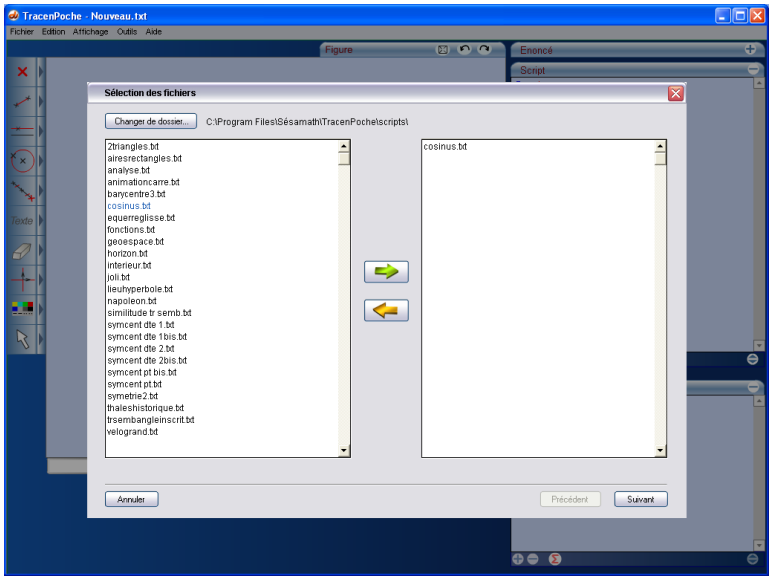
- **Masquer ou afficher les boutons TracenPoche** : comme pour les différentes zones, ce menu permet de choisir quels boutons seront disponibles dans l'interface au chargement de la figure.

d) Le menu Outil

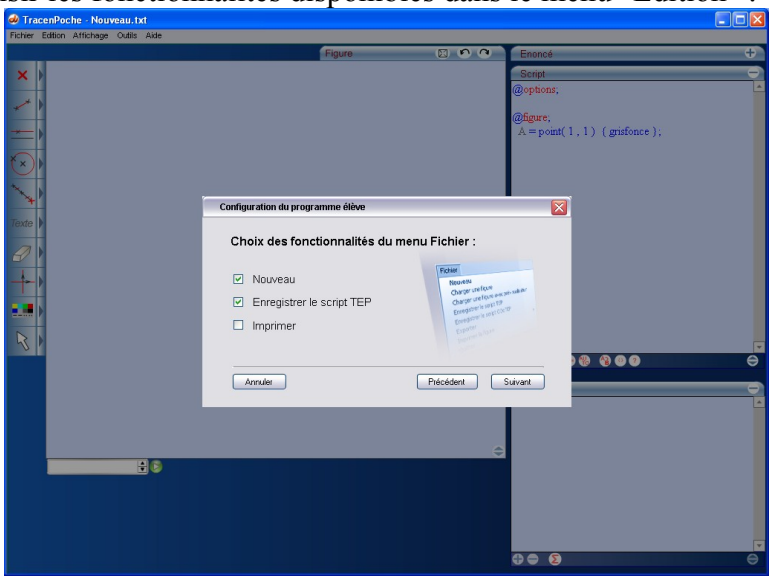


Ce menu permet de créer un environnement TracenPoche, sous la forme d'un exécutable ou d'une page html.
- **Générer une session élève** : permet de créer une version personnalisée du logiciel TracenPoche.

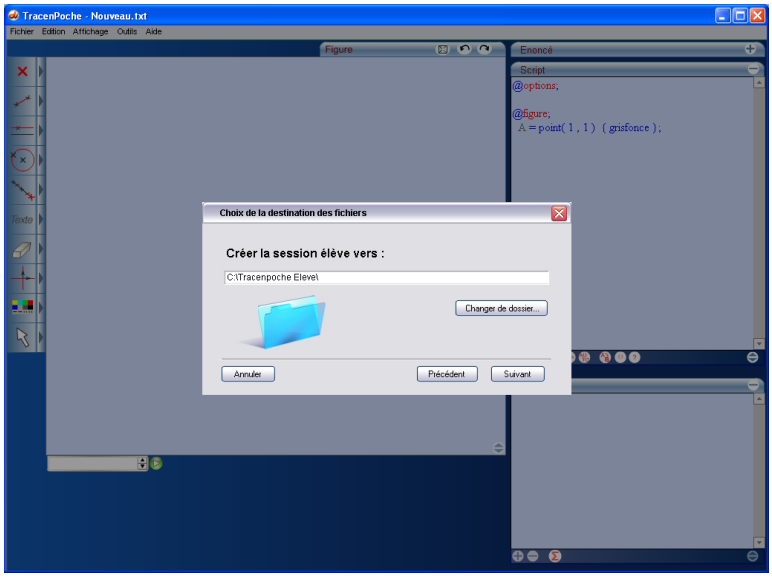
Cette version "élève" de TracenPoche possède des options réduites que le formateur peut personnaliser.
On peut choisir dans un premier temps les scripts mis à la disposition des élèves grâce à la fonction "Charger une figure" :



On peut ensuite choisir les fonctionnalités disponibles dans le menu "Édition" :



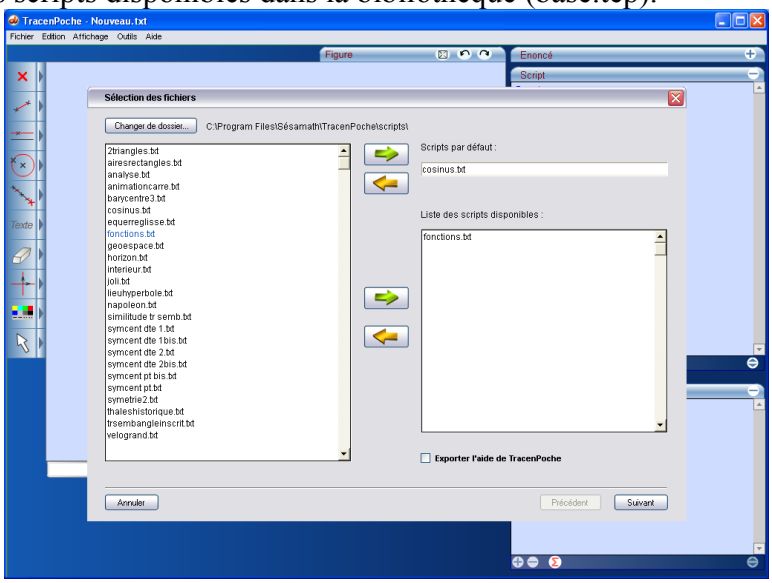
On termine par le choix du répertoire destination des fichiers de la session élève :



Les fichiers obtenus permettent de lancer une version allégée de la version CD de TracenPoche. La barre d'outils ne comporte que les menus Fichier (dont le contenu a été configuré auparavant), Affichage et Aide.

- **Générer une session élève au format HTML** : permet de créer une page html qui contient l'applet TracenPoche ainsi que fichiers de configuration.

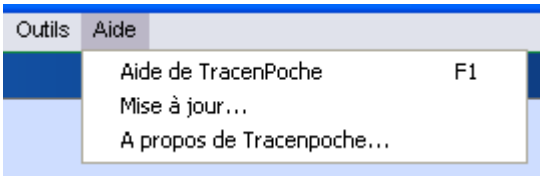
La session élève au format HTML permet de choisir quel script sera chargé à l'ouverture (base.txt) ainsi que les fichiers scripts disponibles dans la bibliothèque (base.tep).



Après avoir spécifié le répertoire destination, 6 fichiers sont générés en dehors des fichiers scripts de la bibliothèque :

- un fichier index.html (la page web en elle-même),
- un fichier tracenpoche.swf (l'applet insérée dans la page web) ,
- un fichier base.txt (il contient le script de la figure par défaut),
- un fichier base.tep (qui référence les scripts de la bibliothèque)
- un fichier macros.txt (qui contient quelques macros TeP)
- un fichier macro.tep (qui contient la référence au fichier macros.txt)

e) Le menu Aide



- **Aide de TracenPoche** : lance l'aide de TracenPoche au format html.
- **Mise à jour ...** : vérifie si une mise à jour du logiciel est disponible. Le programme sera fermé et le travail non sauvegardé sera perdu.
- **A propos de TracenPoche** : les auteurs et les participants au projet TracenPoche.

2.5 Le site : utilisation en ligne

2.5.1 Présentation du site

Le site est accessible à l'adresse www.tracenpoche.net

Ce site est un site de travail pour le développement de TracenPoche. Servant régulièrement à tester les productions, il a vocation à rester en construction car en perpétuelle évolution à l'image du logiciel lui-même.

Il a pour but d'expliquer et de montrer l'utilisation de TracenPoche à 4 niveaux différents :

- pour un utilisateur courant (un élève par exemple),
- pour un utilisateur réalisant des activités (un professeur),
- pour un webmestre qui souhaite réaliser des pages html intégrant des figures interactives (TepWeb),
- pour un programmeur Flash voulant intégrer le module dans sa production (un développeur MathEnPoche avec le module TepNoyau).

Ainsi on y retrouve :

- l'applet en utilisation libre et complète,
- la description de ses composantes graphiques, de son langage script, de son langage d'analyse,
- un tutoriel pour expliquer l'utilisation de l'interface, pas à pas, avec un accès simultané à l'applet pour faire les manipulations.
- une bibliothèque de scripts à voir que l'on peut compléter,
- une page de téléchargement pour utiliser l'applet « ailleurs » : dans un réseau local ou sur un poste isolé,
- et un suivi des révisions du logiciel (important pour découvrir les nouvelles possibilités ou les corrections).

Tout ceci est regroupé dans un menu latéral accessible à tout moment de la visite.

2.5.2 Lancement de TeP en ligne

Le site propose l'applet dans une page PHP (version 3 ou plus) pour pouvoir accéder aux ressources fichiers situées sur le disque dur de l'ordinateur : on peut alors sauvegarder ou ouvrir facilement !

L'applet s'affiche dans une fenêtre surgissante (PopUp) afin de permettre de continuer à naviguer sur le site et de revenir facilement dans cette fenêtre pour utiliser TracenPoche (sous Windows, le basculement peut se faire rapidement au clavier avec la combinaison de touches ALT+TAB).

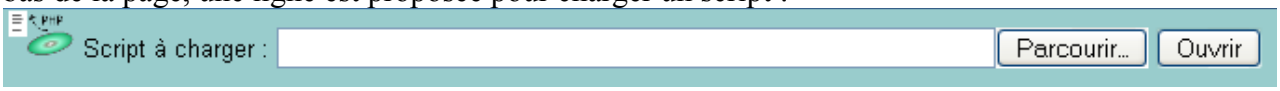
Si cette fenêtre PopUp intitulée "Applet TracenPoche" n'apparaît pas, c'est que le navigateur n'autorise pas les fenêtres PopUp pour des raisons de sécurité élémentaire.
Il faut le lui permettre.
Pour les navigateurs récents (FireFox, IE6 sp2 ...), une barre jaune s'affiche en haut de la fenêtre demandant de passer outre la sécurité si on le souhaite. Les logiciels anti-popups (AntiPopUp, barre Google...) doivent aussi permettre au site TracenPoche d'afficher ce PopUp.

2.5.3 Sauvegarde et chargement

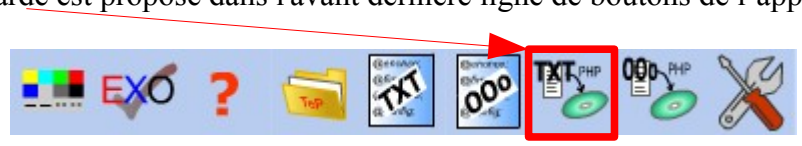
Le lecteur Flash © Macromedia ne permet pas de sauvegarde conviviale sur disque dur, à moins de réaliser des pirouettes pour gérer des sortes de cookies bien connus des internautes mais sans possibilité pour l'utilisateur d'accéder à ses réalisations en dehors de l'applet (la copie de fichiers est quasi impossible) Tout cela pour garantir l'innocuité d'une applet Flash vis à vis de données confidentielles.

On va donc utiliser le fait que l'applet en ligne se lance dans une page avec un serveur PHP.

En bas de la page, une ligne est proposée pour charger un script :

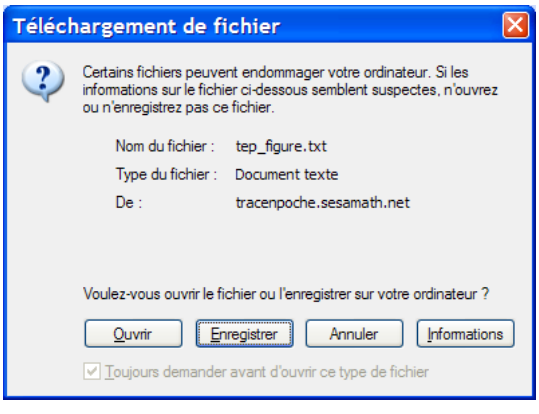


Un bouton de sauvegarde est proposé dans l'avant dernière ligne de boutons de l'applet :

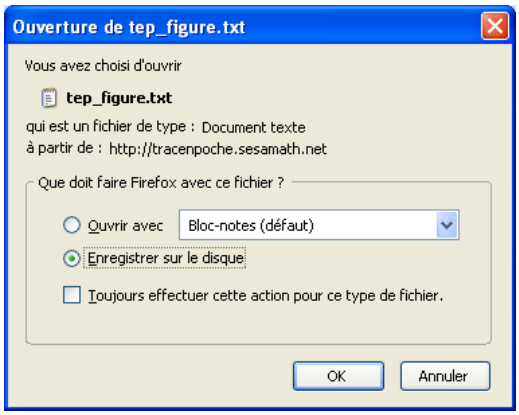


Sauvegarde :

1. Il suffit d'appuyer sur le bouton  de l'applet.
2. Un dialogue habituel apparaît alors (via l'explorateur Internet) et demande confirmation de l'enregistrement sur disque (en rappelant les précautions d'usage) :



Exemple avec MS Interner Explorer



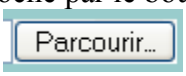
Exemple avec FireFox

3. Appuyer sur Enregistrer ou OK
   Avec MSIE, c'est là qu'il est possible de choisir l'emplacement où mettre le fichier : dossier et nom.

Le script étant un fichier texte, il est conseillé de garder l'extension .txt .
Avec FireFox, c'est l'emplacement et le nom par défaut qui sont utilisés.

Chargement :

1. Localiser le fichier à charger dans TracenPoche par le bouton Parcourir :



   Valider votre choix.
2. Le nom du fichier (avec chemin complet) apparaît dans la zone texte.
3. Appuyer alors sur le bouton Ouvrir :

4. La page se recharge et le script est chargé et affiché.

3. Liste des commandes

3.1 Les options de constructions

Les options des objets permettent :

- de personnaliser l'aspect d'un tracé : couleur, épaisseur, visibilité....
- de personnaliser le comportement d'un objet : point aimanté sur la grille, nombre de chiffres ...
- de créer des animations sur les réels.

Les options s'ajoutent, dans le script, avant le point virgule ; en fin de déclaration de construction, entre accolades { }.

Il ne faut pas confondre avec les options de la figure **`@options;`**

Toutes les options d'apparence peuvent être déclarées manuellement dans le script, à condition d'actualiser la figure après modification du script.

Certaines de ces options (**couleur**, **style de trait**, **aspect d'un point**, **codage** et **caractères**) sont modifiables également grâce au bouton de "mise en forme". Mais, avec la souris, elles sont aussi fonction de l'état de visibilité des objets. Cette visibilité est réglée par des boutons spécifiques : cacher, montrer, rendre visible momentanément/cacher les objets invisibles.

couleur, **style de trait**, **aspect d'un point**, **codage** et **caractères**	**visibilité**
	

La couleur :

- **`blanc, jaune, jauneclair, kakiclair, jaunepaille, rose, magenta`** (uniquement par le script)**`, saumon, orange, rougeclair, rouge, vertclair, vert, vertfonce, kaki, sapin, marron, brique, marronfonce, violetfonce , rougefonce, cyan, bleuciel, bleuocean, bleu (par défaut), bleufonce, violet, gris, grisclair, vertpale, noir`**
- ou codage RVB en hexadécimal selon la syntaxe : **`0xRRVVBB`**

Le style de trait :

- **1** : trait plein par défaut
- **2,3,4** : épaisseur épais, plus épais, très épais
- **6,7 ou 0,8,9** : pointillés petits, simples, longs, plus longs

Ces options s'appliquent aux droites, segments, vecteurs et cercles.

Aspect d'un point :

- **croix1, croix2, croix3** : une croix en 3 tailles (croix2 par défaut)
- **rond1,rond2,rond3** : un "rond" en 3 tailles
- **croix0** ou **rond0** : masque le dessin du point

Visibilité :

- **i, invisible, v, visible** (par défaut) : pour masquer/montrer un objet
- **sansnom** : pour cacher le nom d'un point ou d'une droite

Position :

(x,y) : pour la position d'un nom de point ou d'un texte lié à un point et des entiers / réels où x représente le décalage horizontal et y le décalage vertical en unités du repère (voir page 20).

Codage :

- codage de mesures pour les milieux, segments, médiatrices de segments, images de segments et angles : **/, //, ///, \, \\, \\\, x, o**
- codage de l'angle droit pour médiatrices de segment et perpendiculaires :
  - **q1 ,q2 ,q3 ,q4** pour choisir le quadrant (q1 par défaut), le n° indique les positions successives relatives des quadrants dans le sens direct
  - **q0** pour masquer l'angle droit.

Angles rentrants :

- **r** permet de dessiner l'angle rentrant direct (0° à 360°) et non seulement l'angle saillant associé (0° à 180° direct ou pas).

Remplissage :

- **plein0** ou rien (par défaut), **plein10, plein20, plein30, plein40, plein50 ou plein, plein60, plein70, plein80, plein90, plein100**

Ces options s'appliquent aux objets dont la surface est coloriée (polygones, secteurs, cercles).

Le n° correspond au code alpha d'opacité de 0 à 100 : 0 pour transparent (vide), 100 pour opaque (plein).

Animation :

- **anime** pour les variables entier ou réel .

L'animation d'une variable se fait en ajoutant régulièrement à sa valeur courante la valeur du pas tout en restant entre le minimum et maximum. Une fois le maximum atteint, la valeur repart du minimum.

- **anime1** pour les variables entier ou réel .

Même principe que pour **anime** mais l'animation s'arrête une fois la valeur arrivée à son maximum. Si on relance l'animation, elle repart de son minimum vers son maximum et s'arrête ... etc.

- **oscille** pour les variables entier ou réel .

Même principe que pour **anime** mais une fois arrivée maximum, la valeur décroît régulièrement du pas. Arrivée au minimum, elle croît à nouveau et ainsi de suite.

- **oscille1** pour les variables entier ou réel .

Même principe que pour **oscille** mais l'animation s'arrête une fois la valeur arrivée à son maximum. Si on relance l'animation, elle repart vers son minimum et s'arrête. Si on relance, elle repart vers son maximum et s'arrête ... etc.

- **oscille2** pour les variables entier ou réel .

Même principe que pour **oscille** mais l'animation part vers son maximum puis redescend vers son minimum et s'arrête. Si on relance l'animation, elle repart vers son maximum puis redescend vers son minimum et s'arrête ... etc.

Trace :

- **trace**

Cette option s'applique aux points,droites, segments, vecteurs, demi-droites et cercles.

Lors du déplacement d'un objet de la figure, chaque objet marqué laisse la trace de ses positions précédentes (une alternative rapide aux lieux calculés).

- **pastrace**

Cette option s'applique aux points,droites, segments, vecteurs, demi-droites et cercles.

Caractères :

- **car-4, car-3, car-2, car-1, car+1, car+2, car+3, car+4** pour indiquer de diminuer (-) ou augmenter (+) la taille des caractères d'un texte ou d'un nom de point, droite, demi-droite.
- **gras, italique** pour indiquer de changer le style des caractères d'un texte ou d'un nom de point, droite, demi-droite.

Nombre de décimales :

- **dec0, dec1, dec2 ... dec10** pour indiquer le nombre de décimales à afficher par un objet texte.

Repérage et coordonnées :

- **p1** pour afficher les pointillés figurant les coordonnées du point dans le repère (visible ou pas).

- **coord** ou **coordx** ou **coordy** pour afficher au niveau des axes les 2 coordonnées ou l'abscisse ou l'ordonnée du point dans le repère (visible ou pas).

Mobilité :

- **fixe** pour empêcher le déplacement à la souris d'un point libre ou d'un objet réel ou entier.

- **aimantage** aimante le point sur la grille du repère (même invisible).
- **aimantage5** aimante le point sur les coordonnées multiples de 0.2 (1/5).
- **aimantage10** aimante le point sur les coordonnées multiples de 0.1 (1/10).

- **blinde** pour empêcher la suppression à la souris d'un objet.

Pas à pas :

- **stop** pour marquer les objets balisant les arrêts du pas à pas par blocs (et non objet après objets).

  On active le mode "pas à pas" par la combinaison : bouton souris enfoncé et appui sur la touche P (voir 2.2.3 Le mode pas à pas).

Lieu de point :

- **static** pour qu'un lieu ne soit calculé et affiché qu'une seule fois, la figure est donc plus fluide.

  Il faut actualiser le script si on veut recalculer le lieu après avoir modifié la figure.

3.2 Les mots clés des constructions

- **absc (var)**

Définit la variable contenant l'abscisse d'un "pointsur" un objet (segment, droite, cercle).

Syntaxe :
```
var x = absc(A);
```

Paramètres :
A : "pointsur" un objet

Exemple :
```
M = point( -4.14 , -0.36 );
N = point( 1.91 , 2.76 );
sMN =segment( M , N );
A =pointsur( sMN , 0.39 );
var x =absc(A) { 0.39 };
```

retour

• abscisse (var)

Définit la variable contenant l'abscisse d'un point.

Syntaxe :
```
var x = abscisse(A);
```

Paramètres :
A : point

Exemple :
```
A = point( 3 , 4 );
var x =abscisse(A) { 3 };
```

retour

• aire (var)

Définit la variable contenant l'aire d'un polygone ou d'un disque limité par un cercle.

Syntaxes :
```
var x = aire(A1A2A3...);
```

Paramètres :
A1A2A3... : liste des sommets d'un polygone

retour

• angle (objet)

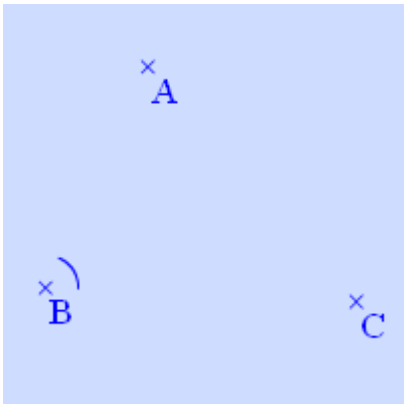
Marque l'angle saillant formé par 3 points.

Syntaxe :
```
a = angle(A,B,C);
```

Paramètres :
A : nom du 1er point
B : nom du 2ème point (sommet de l'angle)
C : nom du 3ème point

Exemple :
```
A = point( 0 , 0.5 );
B = point( -1 , -3 );
C = point( 3 , -3 );
a =angle( A , B , C );
```

Place 3 points A, B, C puis marque l'angle $\widehat{ABC}$ de sommet B



retour

• angle (var)

Définit la variable contenant la mesure d'un angle saillant.

Syntaxe 1 :
```
var x = angle(ABC);
```

Paramètres :
angle saillant non orienté (entre 0° et 180° ou entre 0 rad et pi rad)

Syntaxe 2 :
```
var x = ABC ;
```

Paramètres :
angle orienté saillant (entre -180° et 180° ou entre -pi rad et pi rad)

retour

• anglev (var)

Définit la variable contenant la mesure d'un angle non orienté

Syntaxe :
```
var x = anglev(ABC);
```

Paramètres :
angle non orienté (entre 0° et 360° ou entre 0 rad et 2pi rad)

retour

• arc (objet)

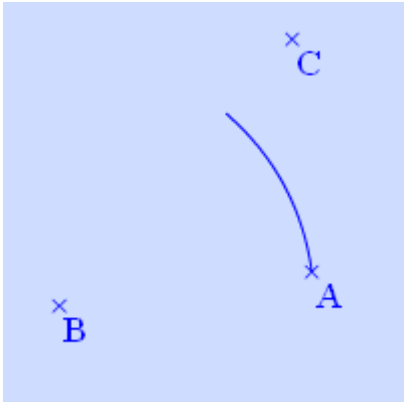
Dessine l'arc basé sur l'angle formé par 3 points.
L'arc est dessiné dans le sens direct à partir du 1er point autour de son centre (le 2ème point) vers le 3ème point.

Syntaxe :
```
a = arc(A,B,C);
```

Paramètres :
A : nom du 1er point qui donne le rayon AB de l'arc
B : nom du 2ème point qui est le centre de l'arc
C : nom du 3ème point qui donne l'ouverture de l'arc

Exemple :
```
A = point( 0 , -2 );
B = point( -3 , -2 );
C = point( 0.5 , 1 );
a =arc( A , B , C );
```

Place 3 points A,B,C et dessine l'arc a partant de A vers C, de centre B.



retour

• arc (var)

Définit la variable contenant la mesure d'un arc.

Syntaxe :
```
var x = arc(ABC);
```

Paramètres :
A : nom du 1er point qui donne le rayon AB de l'arc
B : nom du 2ème point qui est le centre de l'arc
C : nom du 3ème point qui donne l'ouverture de l'arc

retour

• barycentre

Place le barycentre d'un ensemble de points affectés d'une masse.

Syntaxe :

```
B=barycentre(A1,m1,A2,m2,...);
```

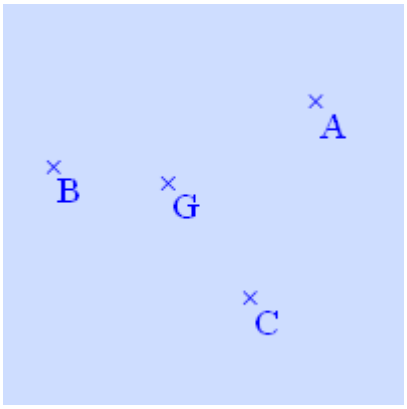
Paramètres :

B : nom du point obtenu
A1 : 1er point
m1 : masse du 1er point
A2 : 2ème point
m2 : masse du 2ème point
...
Le nombre d'objets n'est pas limité

Exemple :

```
  A = point( 2 , 1 );
  B = point( -2 , 0 );
  C = point( 1 , -2 );
  G =barycentre( A , 1 , B , 2 , C ,
1  );
```

Place 3 points A, B, C puis le barycentre de l'ensemble { (A,1), (B,2), (C,1) }.



retour

• bissectrice

Trace la bissectrice de l'angle formé par 3 points (dans le sens direct)

Syntaxe :

```
d=bissectrice(A,B,C);
```

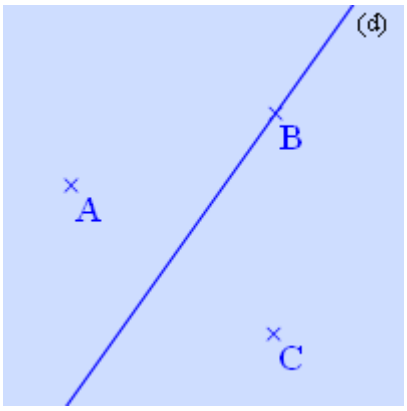
Paramètres :

A : nom du 1er point
B : nom du 2ème point (sommet de l'angle)
C : nom du 3ème point

Exemple :

```
  A = point( 1 , 6 );
  B = point( 4 , 7 );
  C = point( 4 , 3 );
  d =bissectrice( A , B , C );
```

Place 3 points A, B, C puis trace la bissectrice (d) de l'angle $\widehat{ABC}$ de sommet B.



retour

• cercle

Trace un cercle de centre un point et passant par un second point.

Syntaxe :

```
ce=cercle(A,B);
```

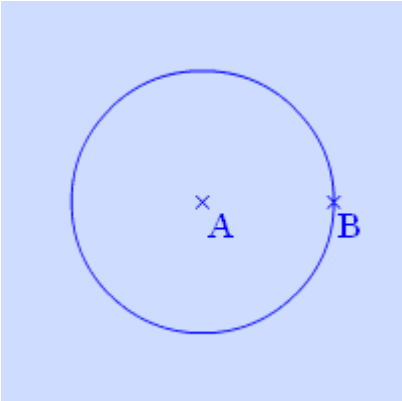
Paramètres :

A : nom du point qui est le centre du cercle
B : nom du point par lequel passe le cercle

Exemple :

```
A = point( 0 , 0 );
B = point( 2 , 0 );
ce =cercle( A , B );
```

Place 2 points A et B puis trace le cercle de centre A passant par B.



retour

- **cercleDia**

Trace un cercle défini par un diamètre.

Syntaxe :

```
ce=cercle(A,B);
```

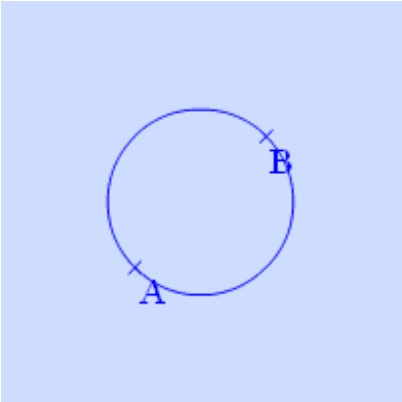
Paramètres :

A : nom d'une extrémité du diamètre
B : nom de l'autre extrémité du diamètre

Exemple :

```
A = point( -1 , -1 );
B = point( 1 , 1 );
ce =cercledia( A , B );
```

Place 2 points A et B puis trace le cercle de diamètre [AB].



retour

- **cercleRayon**

Trace un cercle défini par son centre et une valeur de son rayon.

Syntaxe :

```
ce=cercle(A,x);
```

Paramètres :

A : nom du point centre du cercle
x : valeur numérique du rayon ou objet numérique (**entier**, **réel** ou **var**)

Exemple :

```
A = point( 0 , 0 );
ce =cerclerayon( A , 2 );
```

Place 1 point A puis trace le cercle de centre A et de rayon 2 (unités du repère).

retour

• commentaire //

Indique que la ligne se terminant par un ; est un commentaire : son contenu n'est pas interprété mais juste affiché dans le script.
Un commentaire doit se terminer par un point-virgule ; et ne peut donc pas en contenir !

Syntaxe :

```
// mon commentaire ;
```

Exemple :

```
// je place d'abord un point A ;
A =point(2,2);
```

Pose un commentaire dans le script puis place un point A

retour

• demidroite

Trace la demi-droite définie par 2 points

Syntaxe :

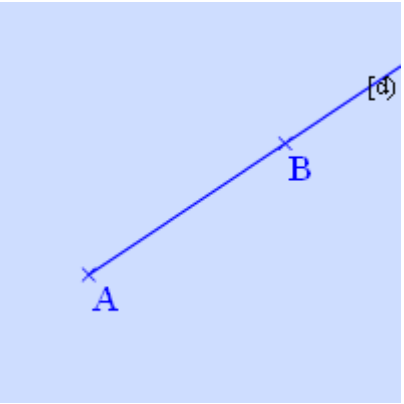
```
d = demidroite(A,B);
```

Paramètres :
A : nom du 1er point, origine de la demi-droite
B : nom du 2nd point

Exemple :

```
A = point( -2 , -1 );
B = point( 1 , 1 );
d =demidroite( A , B );
```

Place 2 points A et B puis trace la demi-droite nommée d passant par A et B : [AB).



retour

• distance (var)

Définit la variable contenant la distance entre 2 points.

Syntaxe :

```
var x = AB;
```

Paramètres :

A : nom du 1er point
B : nom du 2nd point

retour

• droite

Trace la droite passant par 2 points.

Syntaxe :

```
d = droite(A,B);
```

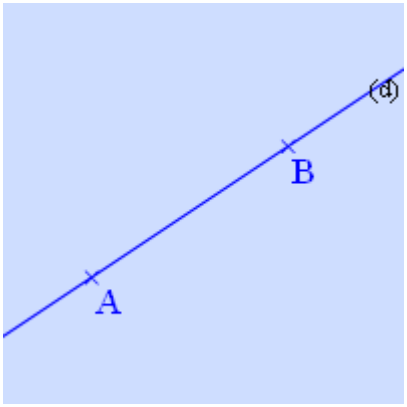
Paramètres :

A : nom du 1er point
B : nom du 2nd point

Exemple :

```
A = point( -2 , -1 );
B = point( 1 , 1 );
d =droite( A , B );
```

Place 2 points A et B puis trace la droite (d) passant par A et B.



retour

• droiteEqr

Trace la droite définie par son équation réduite cartésienne dans le repère actuel (visible ou pas).

Syntaxe :

```
d = droiteeqr(a,b);
```

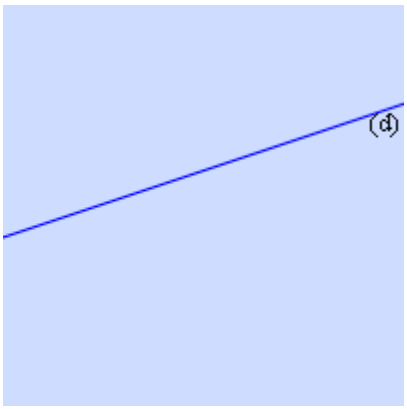
Paramètres :

a,b : les coefficients de l'équation dans sa forme [y=ax+b]
Ils peuvent être écrits sous forme de fraction : 1/2, 3/4 ...

Exemple :

```
d =droiteeqr( 1/3 , 1 );
```

Trace la droite (d) d'équation [$y = \frac{x}{3} + 1$]



retour

• droiteEq

Trace la droite définie par son équation cartésienne dans le repère actuel (visible ou pas).

Syntaxe :

```
d = droiteeq(a,b,c);
```

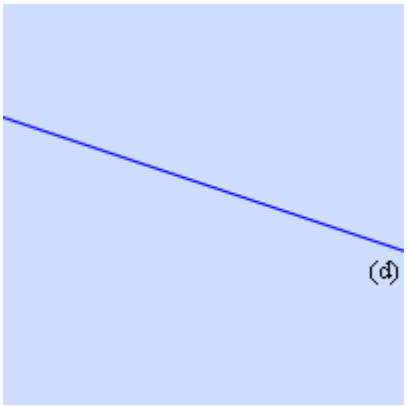
Paramètres :

a,b,c : les coefficients de l'équation dans sa forme [ax+by+c=0]

Exemple :

```
d =droiteeq( 1 , 3 , 3 );
```

Trace la droite (d) d'équation x+3y+3=0



retour

• droitev

Trace la droite définie par un point et un vecteur directeur.

Syntaxe :

```
d = droitev(A,v);
```

Paramètres :

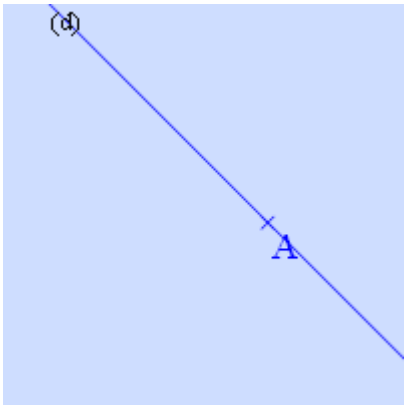
A : nom du point

v : nom du vecteur directeur

Exemple :

```
A = point( 0 , 5 );
v =vecteurcoord( -1 , 1 );
d =droitev( A , v );
```

Place un point A, définit le vecteur $\vec{v}$ de coordonnées (-1,1) puis trace la droite passant par A de vecteur directeur $\vec{v}$.



retour

• entier

Définit une variable contenant une valeur entière.

Syntaxe :
```
a = entier(v,min,max,pas);
```

Paramètres :
v : valeur initiale
min : valeur minimale possible (souris)
max : valeur maximale possible (souris)
pas : valeur à ajouter pour passer d'une valeur à une autre valeur (souris)

Le déplacement horizontal de la **souris**, bouton enfoncé sur le point rouge au-dessus de l'objet dessiné, permet de faire varier la valeur de l'entier **a** entre les 2 bornes **min** et **max**, avec des sauts de **pas** unités depuis v.

retour

• fonction

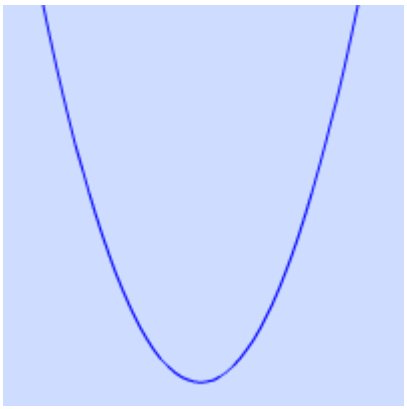
Définit une fonction et en trace la courbe représentative dans le repère actuel (visible ou pas).

Syntaxe :
```
f = fonction(formule);
```

Paramètres :
formule : expression de la fonction avec comme variable x.

Exemple :
```
f =fonction( x^2 );
```

Dessine la courbe de la fonction carrée [$y=x^2$] dans le repère courant.



retour

• fonction µ

Permet de conditionner l'existence d'un objet grâce à un objet varsi.

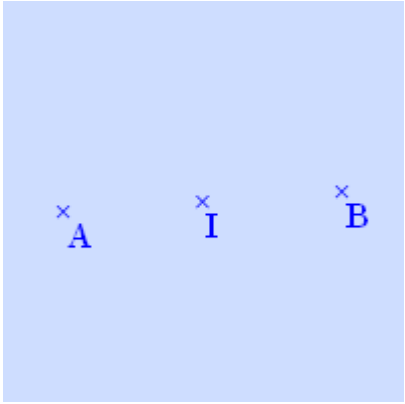
Syntaxe :
```
nom =  µ(z) typeobjet(paramètres);
```

Paramètres :
z : varsi dont l'une des 2 valeurs prise est 0

Exemple :

```
A = point( -3 , 0 );
B = point( 1.6 , 0.3 );
varsi z =[AB>4,1,0];
I = μ(z) milieu( A , B );
```

Le point I, milieu de du segment [AB], n'est construit que si AB>4.



retour

- **groupe**

Regroupe dans un ensemble certains objets afin de travailler facilement dessus.

Syntaxe :

```
g = groupe(o1,o2,o3,...);
```

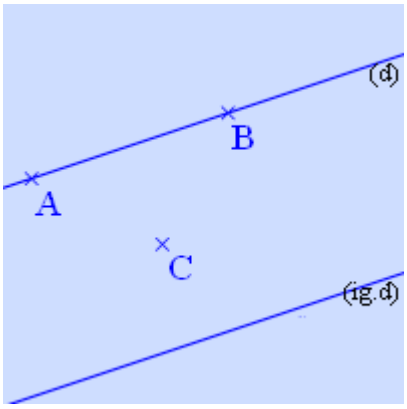
Paramètres :

o1,o2,o3,... : liste des objets à grouper, séparés par une virgule.
Le nombre d'objets n'est pas limité et leurs natures peuvent être différentes (points avec droites, ...).

Exemple :

```
A = point( -2 , 1 );
B = point( 1 , 2 );
C = point( 0 , 0 );
d =droite( A , B );
g =groupe( A , d );
sC =symetrie( C );
ig =image( sC , g );
```

Place 3 points A, B et C puis trace la droite (AB) nommée d puis regroupe le point A et la droite (d) dans le groupe g.
Définie la symétrie sC de centre C et nomme ig le groupe formé par les images par la symétrie sC des éléments du groupe g.
Cela évite de définir séparément les images de A et d : ici on a ig.A image de A et ig.d image de d !
Pour le moment on ne peut pas intervenir sur la mise en forme des images.



retour

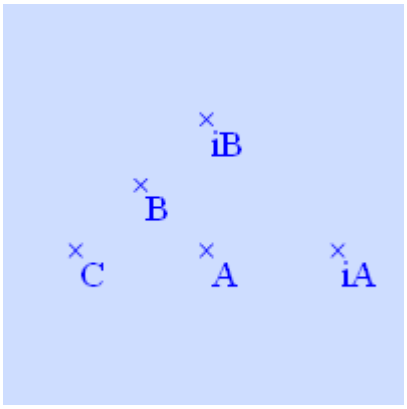
- **homotethie**

Définit une homothétie par son centre et son rapport.
Pour construire une image, il faut utiliser **image**.

Syntaxe :
```
h = homothetie(O,k);
```

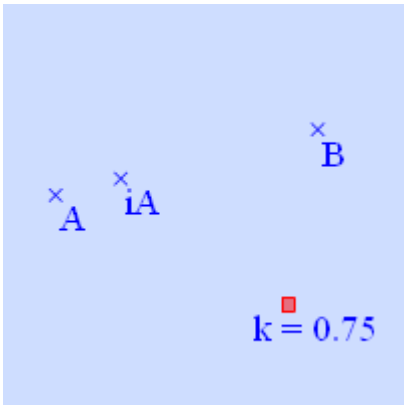
Paramètres :
O : nom du centre
k : valeur numérique du rapport ou objet numérique (**entier**, **réel** ou **var**) valeur du rapport

Exemple 1 :
```
A = point( 0 , -1 );
B = point( -1 , 0 );
C = point( -2 , -1 );
h =homothetie( C , 2 );
iA =image( h , A );
iB =image( h , B );
```



Place 3 points A,B,C et définit l'homothétie h de centre A et de rapport 2
puis construit l'image iA de A par l'homothétie h et l'image iB de B par l'homothétie h.

Exemple 2 :
```
A = point( -2 , 0 );
B = point( 2 , 1 );
k =reel( 0.75 , 0.5 , 2 , 0.1 )  { (1.46,-1.67) };
h =homothetie( B , k );
iA =image( h , A );
```



Place 2 points A, B
puis définit la variable k, valant 0,75 au départ, variant entre 0,5 et 2 de 0,1 en 0,1,
puis définit l'homothétie h de centre A et de rapport k
puis construit l'image iA de A par l'homothétie h.

retour

• image

Construit l'image d'un objet en utilisation une transformation du plan définie au préalable : symétrie, réflexion, translation, rotation, homothétie, similitude.

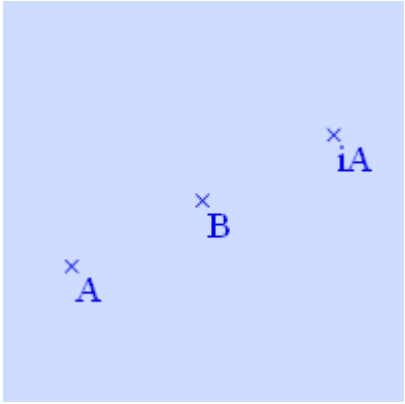
Syntaxe :
```
iM = image(t,M);
```

Paramètres :
t : nom de la transformation
M : nom du point, de la droite ... ou du groupe dont on construit l'image

Exemple :

```
A = point( -2 , 0 );
B = point( 0 , 1 );
sB =symetrie( B );
iA =image( sB , A );
```

Place 2 points A,B et définit la symétrie sB de centre B, puis construit l'image iA de A par la symétrie sB.



retour

• intersection

Construit l'intersection de 2 objets (de type segment, droite, demi-droite, cercle, arcs uniquement).
Dans le cas de 2 droites/demi-droites ou segments, c'est simple : un seul point peut exister..
Dans le cas où un cercle ou un arc entre en jeu, il y a au plus 2 points possibles à construire, il faut
- soit indiquer à TracenPoche lequel choisir pour définir le point à construire/dessiner quand il existe : le 1er ou le 2ème dans l'ordre du calcul par TracenPoche (choix).
- soit indiquer un point à éviter. Dans ce cas, si les 2 points d'intersections restent possibles, le 1er calculé par TracenPoche sera utilisé.

Syntaxe 1 :

```
M = intersection(d1,d2);
```

Paramètres :
d1: nom de la 1ère droite/segment
d2 : nom de la 2ème droite/segment

Syntaxe 2 :

```
M = intersection(l1,c2,i);
```

Paramètres :
l1 : nom du 1er objet : cercle, droite ou segment !
c2 : nom du cercle
i : 1 ou 2 : choix de l'un des 2 points possibles

Syntaxe 3 :

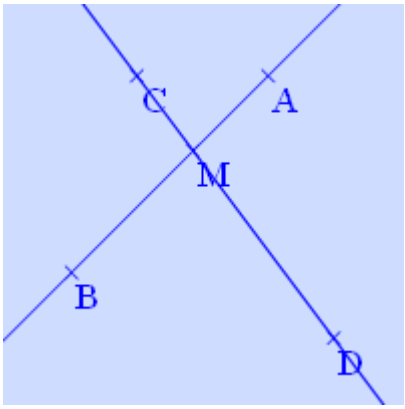
```
M = intersection(l1,c2,P);
```

Paramètres :
l1 : nom du 1er objet : cercle, droite ou segment !
c2 : nom du cercle
P : nom du point à éviter

Exemple 1 :

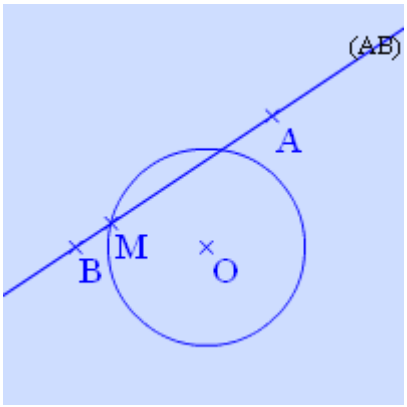
```
A = point( 1 , 2 );
B = point( -2 , -1 );
AB =droite( A , B );
C = point( -1 , 2 );
D = point( 2 , -2 );
CD =droite( C , D );
M =intersection( AB , CD );
```

Trace 2 droites l'une passant par A et B nommée AB et l'autre passant par C et D nommée CD puis construit le point M intersection des 2 droites.



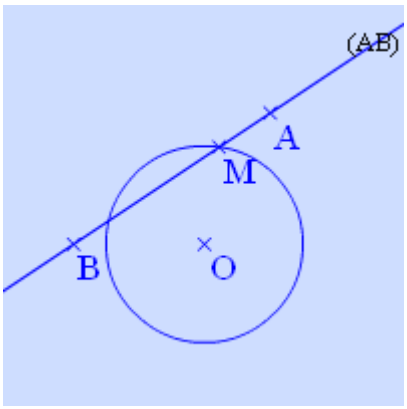
Exemple 2 :

```
A = point( 1 , 2 );
B = point( -2 , 0 );
AB =droite( A , B );
O = point( 0 , 0 );
ce =cerclerayon( O , 1.5 );
M =intersection( AB , ce , 1 );
```



Trace la droite passant par A et B et un cercle de centre un point O et de rayon 1.5 puis construit le point M comme l'une des 2 intersections du cercle et de la droite (ici la 1ère dans le calcul de TracenPoche : 1)

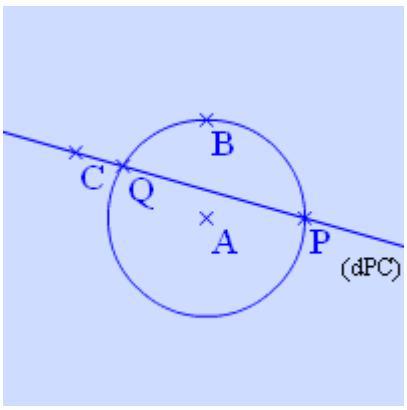
En changeant le 1 par 2 on aurait obtenu l'autre point.




```
A = point( 1 , 2 );
B = point( -2 , 0 );
AB =droite( A , B );
O = point( 0 , 0 );
ce =cerclerayon( O , 1.5 );
M =intersection( AB , ce , 2 );
```

Exemple 3 :

```
A = point( 0 , 0 );
B = point( 0 , 1.5 );
ceAB =cercle( A , B );
C = point( -2 , 1 );
P =pointsur( ceAB , 0 );
dPC =droite( P , C );
Q =intersection( dPC , ceAB , P );
```



Trace un cercle de centre A passant par B.
Place un point C sur le plan, puis P un point sur le cercle.
La droite (PC) recoupe le cercle en Q : Q est le point d'intersection qui n'est pas le point P !

retour

• lieu

Construit le lieu d'un point dépendant d'un point sur (**pointsur**) un objet.

Syntaxe 1 :

```
l = lieu (M,H);
```

Paramètres :
M : point dont on veut le lieu
H : point sur un objet

Syntaxe 2 :

```
l = lieu (M,H,n);
```

Paramètres :
M : point dont on veut le lieu
H : point sur un objet
n : valeur indiquant le nombre de points calculés, 21 par défaut, plus il grand plus c'est lent !

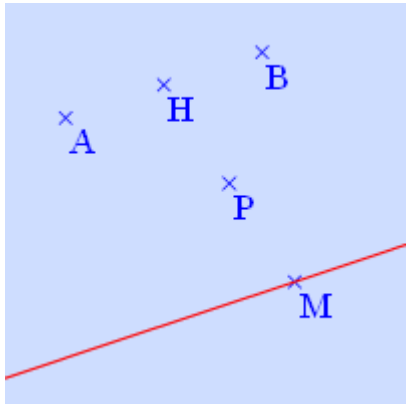
Syntaxe 3 :

```
l = lieu (M,H,n,min,max);
```

Paramètres :
M : point dont on veut le lieu
H : point sur un objet
n : valeur indiquant le nombre de points calculés, 21 par défaut, plus il grand plus c'est lent !
min,max : valeurs mini et maxi entre lesquelles varient la position de H
(dans le repère de l'objet sur lequel il est posé).
Par défaut :
- pour une droite : -10,10 (unités)
- pour un cercle : -180, +180 (degrés)
- pour un segment ou un arc : 0,1 (unités)

Exemple 1 :

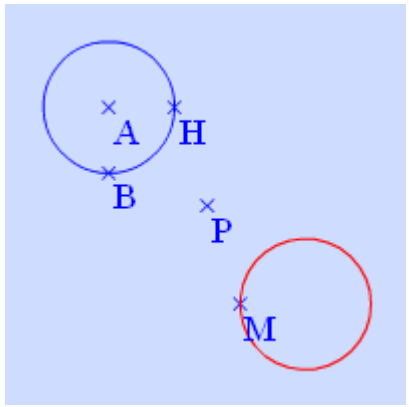
```
A = point( -2 , 0 );
B = point( 1 , 1 );
H =pointsur( A , B , 0.5 );
P = point( 0.5 , -1 );
M =symetrique( H , P );
l =lieu( M , H )  { rouge };
```



Place 2 points A et B, puis le point H sur (AB) à la position 0,5 (milieu de [AB]).
Place un point P puis construit le symétrique M de H par rapport à P.
Dessine en rouge le lieu l du point M quand H varie sur (AB).

Exemple 2 :

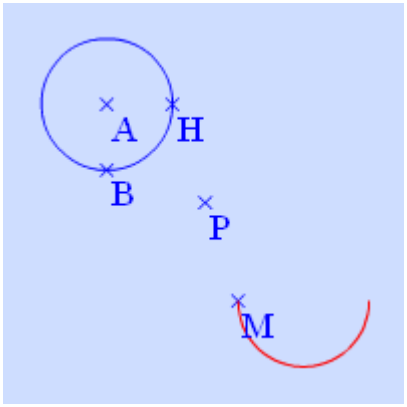
```
A = point( -1 , 1 );
B = point( -1 , 0 );
c1 =cercle( A , B );
H =pointsur( c1 , 0 );
P = point( 0.5 , -0.5 );
M =symetrique( H , P );
l =lieu( M , H , 31 )  { rouge };
```



Place 2 points A et B,
puis trace le cercle (c1) de centre A passant par B
puis place le point H sur (AB) à la position 0°.
Place un point P puis construit le symétrique M de H par rapport à P.
Dessine le lieu l du point M quand H varie sur le cercle (c1) avec 31 points calculés.

Exemple 3 :

```
  A = point( -1 , 1 );
  B = point( -1 , 0 );
  c1 =cercle( A , B );
  H =pointsur( c1 , 0 );
  P = point( 0.5 , -0.5 );
  M =symetrique( H , P );
  l =lieu( M , H , 31 , 0 , 180 )
{ rouge };
```



Place 2 points A et B,
puis trace le cercle (c1) de centre A passant par B
puis place le point H sur (AB) à la position 0°.
Place un point P puis construit le symétrique M de H par rapport à P.
Dessine le lieu l du point M quand H varie sur le cercle (c1) avec 31 points calculés, le point H restant dans le demi-cercle supérieur : arc définit par les angles allant de 0° à 180°.

retour

• max (var)

Définit la variable contenant le maximum d'une liste de nombres.

Syntaxe :

```
var x = max(y,z,...);
```

Paramètres :
y, z, ... : valeurs numériques et/ou objets numériques

retour

• mediatrice

Trace la médiatrice d'un segment.

Syntaxe 1 :

```
d = mediatrice(A,B);
```

Paramètres :
A : nom d'une extrémité du segment
B : nom de l'autre extrémité
Pas de codage.

Syntaxe 2 :

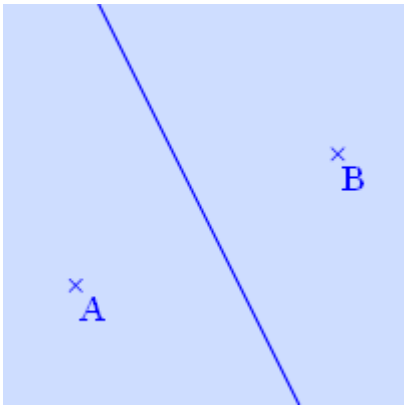
```
d = mediatrice(s1);
```

Paramètres :
d : nom de la médiatrice
s1 : nom du segment
L'angle droit est codé par défaut.
Le milieu est codé sur demande.

Exemple 1 :

```
A = point( -2 , 0 );
B = point( 2 , 2 );
d =mediatrice( A , B );
```

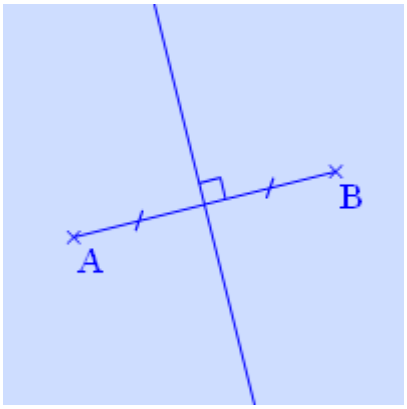
Place 2 points A et B puis trace la médiatrice de [AB].



Exemple 2 :

```
A = point( -2 , 0 );
B = point( 2 , 1 );
s1 =segment( A , B );
d =mediatrice( s1 )  { / };
```

Place 2 points A et B puis trace le segment [AB] et marque la médiatrice du segment [AB] avec le codage de l'angle droit et du milieu.



retour

- **milieu**

Trace le milieu d'un segment.

Syntaxe 1 :

```
M = milieu(A,B);
```

Paramètres :
A : nom d'une extrémité du segment
B : nom de l'autre extrémité

Syntaxe 2 :

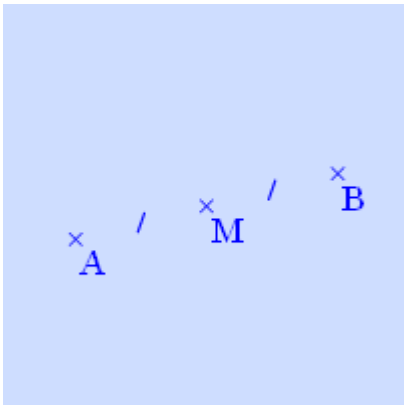
```
M = milieu(s1);
```

Paramètres :
s1 : nom du segment

Exemple 1 :

```
A = point( -2 , 0 );
B = point( 2 , 1 );
M =milieu( A , B )  { / };
```

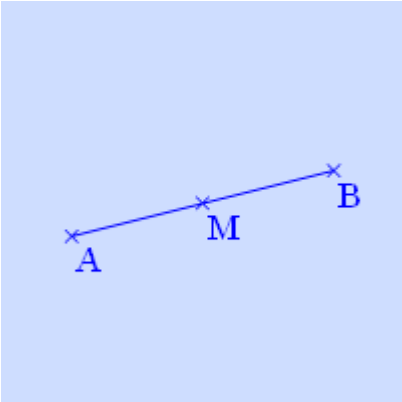
Place 2 points A et B puis place le milieu M de [AB] codé par 2 /.



Exemple 2 :

```
A = point( -2 , 0 );
B = point( 2 , 1 );
s1 =segment( A , B );
M =milieu( s1 );
```

Place 2 points A et B puis trace le segment [AB] et place le milieu M du segment.



retour

- **min(var)**

Définit la variable contenant le minimum d'une liste de nombres.

Syntaxe :

```
var x = min(y,z,...);
```

Paramètres :

y, z, ... : valeurs numériques et/ou objets numériques

retour

- **ordonnee (var)**

Définit la variable contenant l'ordonnée d'un point.

Syntaxe :

```
var x = ordonnee(A);
```

Paramètres :

A : point

Exemple :

```
A = point( 3 , 4 );
var x =ordonnee(A) { 4 };
```

retour

- **parallèle**

Trace la droite passant par un point et parallèle à une droite ou un segment.

Syntaxe :

```
d1 = parallele(A,d);
```

ou

```
d1 = parallele(d,A);
```

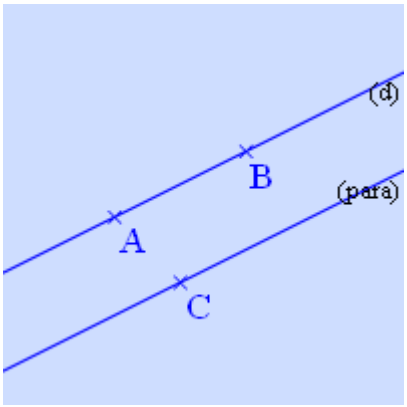
Paramètres :

A : nom du point

d : nom de la droite à laquelle d1 sera parallèle

Exemple :

```
A = point( -1 , 0 );
B = point( 1 , 1 );
C = point( 0 , -1 );
d =droite( A , B );
para =parallele( C , d );
```



Place 3 points A,B,C puis trace la droite (d) passant par A et B, et construit la droite (para) passant par C et parallèle à (d).

retour

• perpendiculaire

Trace la droite passant par un point et perpendiculaire à une droite ou un segment.

Syntaxe :

```
d1 = perpendiculaire(A,d);
```

ou

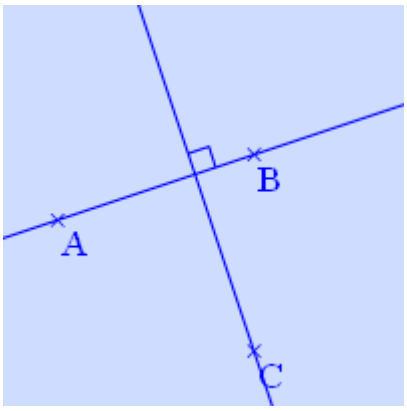
```
d1 = perpendiculaire(d,A);
```

Paramètres :

A : nom du point
d : nom de la droite à laquelle d1 sera perpendiculaire

Exemple :

```
A = point( -2 , 0 );
B = point( 1 , 1 );
C = point( 1 , -2 );
d =droite( A , B );
perp =perpendiculaire( C , d );
```



Place 3 points A,B,C puis trace la droite (d) passant par A et B, et construit la droite (perp) passant par C et perpendiculaire à (d).

retour

• perimetre (var)

Définit la variable contenant le périmètre d'un polygone.

Syntaxe :

```
var x = perimetre(ABC);
```

Paramètres :

retour

• point

Place un point défini par ses coordonnées dans le repère actuel (visible ou pas).

Syntaxe 1 :

```
A = point(x,y);
```

Paramètres :

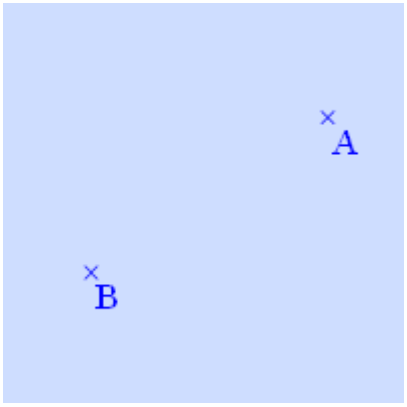
x : abscisse du point
y : ordonnée du point

Syntaxe 2 :
```
A = point();
```

Paramètres :
Le point est placé aléatoirement (au hasard) dans le repère

Exemple :
```
A = point( 2 , 1 );
B = point();
```

Place un point A de coordonnées (2, 1) et place un point B au hasard.



retour

- **pointAimante**

Définit un point aimanté par un objet (cercle, droite ou point).

Syntaxe 1 :
```
M = pointaimante(a,b,MA = k_x%);
```
Paramètres :
a : abscisse de M
b : ordonnée de M
A : point de la figure
k : valeur numérique ou objet numérique
x : entier

Syntaxe 2 :
```
M = pointaimante(a,b,M appartient
d_x%);
```
Paramètres :
a : abscisse de M
b : ordonnée de M
d : droite
x : entier

Syntaxe 3 :
```
M = pointaimante(a,b,M appartient
AB_x%);
```
Paramètres :
a : abscisse de M
b : ordonnée de M
A, B : points de la figure
x : entier

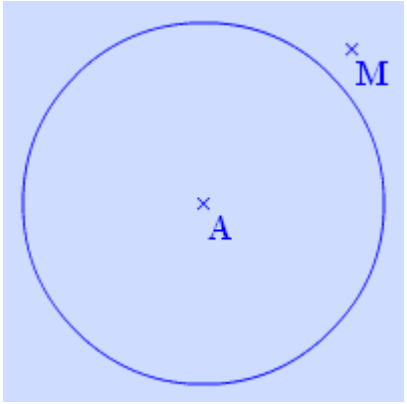
Syntaxe 4 :
```
M = pointaimante(a,b,M sur A_x%);
```
Paramètres :
a : abscisse de M
b : ordonnée de M
A : point de la figure
x : entier

Syntaxe 5 :
```
M = pointaimante(a,b,M sur
(k1,k2)_x%);
```
Paramètres :
a : abscisse de M
b : ordonnée de M
k1, k2 : valeurs numériques ou objets numériques
x : entier

Exemple 1 :

```
A = point( -1.5 , 0 );
cerayA3 = cerclerayon( A , 3 );
M = pointaimante( 1 , 3 ,MA=3_10% );
```

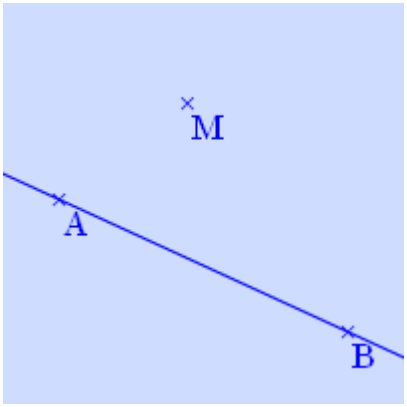
M sera aimanté sur le cercle de centre A et de rayon 3 dès que la longueur MA sera égale à 3 à 0,1 près.



Exemple 2 :

```
A = point( -1 , 0 );
B = point( 4 , -1 );
d = droite( A , B );
M = pointaimante( 1 , 2 , M
appartient AB_20% );
```

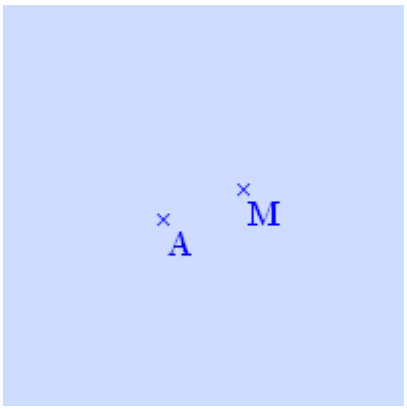
M sera aimanté sur la droite d dès que la distance entre M et la droite d sera inférieure ou égale à 0,2



Exemple 3 :

```
A = point( -1 , 0 );
M = pointaimante( 1 , 2 , M sur A_5%
);
```

M sera aimanté sur le point A dès qu'il sera à une distance de A inférieure ou égale à 0,05



retour

- **pointSur**

Définit un point semi libre placé sur un objet (segment, droite, cercle, arc, ...).

Syntaxe 1 :

```
M = pointsur(A,B,x);
```

Paramètres :

A : nom du point origine
B : nom du point directeur
x : valeur numérique ou objet numérique (**entier**, **réel**,**var**,...) abscisse du point M sur l'axe gradué défini par A et B

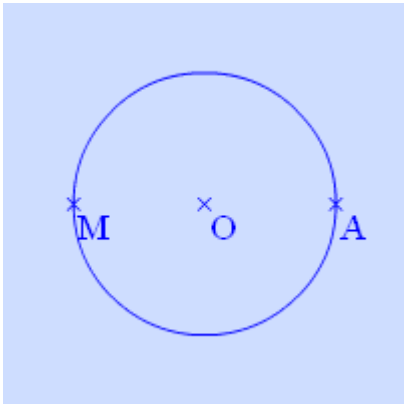
Syntaxe 2 :
```
M = pointsur(d,x);
```

Paramètres :
d : nom d'une droite, demi-droite ou d'un segment voire d'une courbe de fonction
x : valeur numérique ou objet numérique (**entier**, **réel**,**var**,...) égale à l'abscisse du point M dans le repère général (si d est vertical, ce sera son ordonnée)

Syntaxe 3 :
```
M = pointsur(ce,x);
```

Paramètres :
ce : nom d'un cercle ou d'un arc
x : valeur numérique ou objet numérique (**entier**, **réel**,**var**,...) égale à la mesure de l'angle orienté en degrés déterminant la position du point M par rapport à la demi-droite partant du centre du cercle et horizontale
si x=0, M est à droite du centre sur le cercle
si x=180, M est à gauche du centre sur le cercle
si x=90, M est au dessus du centre sur le cercle

Exemple :
```
O = point( 0 , 0 );
A = point( 2 , 0 );
Ce = cercle( O , A );
M = pointsur( Ce , 180 );
```



Place un point O et un point A (horizontalement à sa droite),
dessine le cercle de centre O passant par A,
place M sur le cercle de sorte que l'angle $\widehat{AOM}$ fait 180°.

retour

- **polygone**

Trace un polygone formé par une liste de sommets (points).

Syntaxe 1 :
```
p = polygone(A1,A2,A3,...);
```

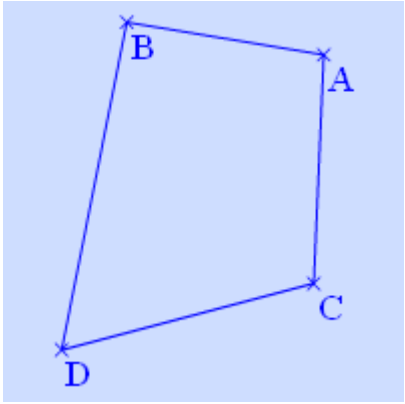
Paramètres :
A1,A2,A3,... : liste des sommets séparés par une virgule.
Le nombre de sommets n'est pas limité.

Syntaxe 2 :
```
p = polygone(g);
```

Paramètres :
g : groupe de points

Exemple 1 :

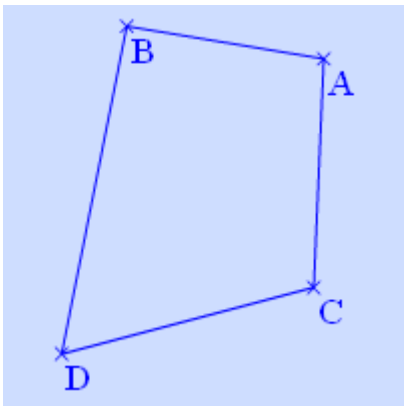
```
A = point( 2 , 3.5 );
B = point( -1 , 4 );
C = point( 2 , 0 );
D = point( -2 , -1 );
p =polygone( A , B , D , C );
```



Place 4 points A, B,C et D puis trace le polygone ABDC nommé p.

Exemple 2 :

```
A = point( 2 , 3.5 );
B = point( -1 , 4 );
C = point( 2 , 0 );
D = point( -2 , -1 );
g = groupe( A , B , D , C );
p =polygone( g );
```



On obtient la même configuration qu'au-dessus : place 4 points A, B,C et D puis les rassemble dans le groupe g
et trace le polygone p avec les points de g (dans l'ordre).

retour

- **projete**

Trace le projeté d'un point sur une droite ou un segment

Syntaxe 1 :

```
M = projete(A,d);
```

Paramètres :
A : point à projeter
d : droite ou segment sur lequel on projette PERPENDICULAIREMENT

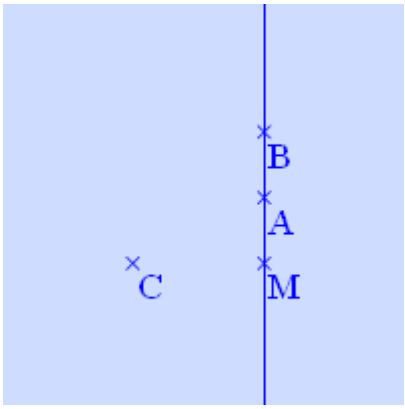
Syntaxe 2 :

```
M = projete(A,d,d1);
```

Paramètres :
A : point à projeter
d : droite ou segment sur lequel on projette
d1 : droite ou segment PARALLELEMENT auquel on projette

Exemple 1 :

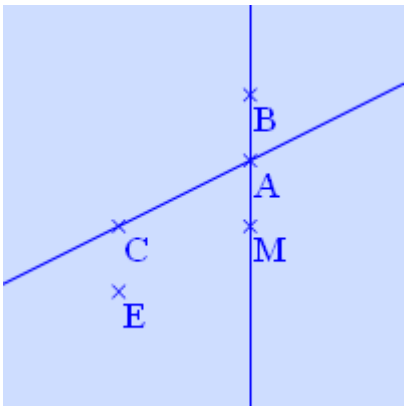
```
A = point( 1 , 1 );
B = point( 1 , 2 );
C = point( -1 , 0 );
d =droite( A , B );
M =projete( C , d );
```



Place 3 points A,B,C et trace la droite (AB) nommée d, puis trace le projeté orthogonal de C sur d.

Exemple 2 :

```
A = point( 1 , 1 );
B = point( 1 , 2 );
C = point( -1 , 0 );
d =droite( A , B );
d1 =droite( A , C );
E = point( -1 , -1 );
M =projete( E , d , d1 );
```



Place 3 points A, B, C puis trace les droites (AB) et (AC) nommées d et d1
puis, place un point E et construit son projeté M sur d parallèlement à d1.

retour

• random (var)

Définit la variable contenant un nombre choisi de manière aléatoire.

Syntaxe 1:

```
var x = random();
```

Renvoie un réel de l'intervalle[0;1[(0 inclus, 1 exclus)

Syntaxe 2 :

```
var x =random(y,z);
```

Paramètres :
y, z : entiers (avec y < z)

Renvoie un entier de l'intervalle [y, z]

Syntaxe 3 :

```
var x =random(y);
```

Paramètres :
y : entier

Renvoie un entier de l'intervalle [0, y]

retour

• reel

Définit une variable contenant une valeur réelle.

Syntaxe :

```
a = reel(v,min,max,pas);
```

Paramètres :
v : valeur initiale
min : valeur minimale possible (souris)
max : valeur maximale possible (souris)
pas : valeur à ajouter pour passer d'une à une autre valeur (souris)

Le déplacement horizontal de la **souris**, bouton enfoncé sur le point rouge au-dessus de l'objet dessiné, permet de faire varier la valeur du réel **a** entre les 2 bornes **min** et **max**, avec des sauts de **pas** unités depuis v.

retour

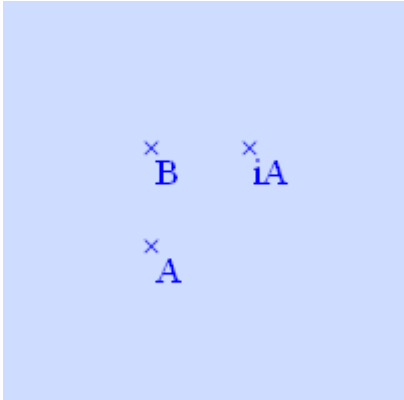
• rotation

Définit une rotation par son centre et son angle dans le sens direct du repère. Par défaut l'angle est en degrés.
Pour construire une image, il faut utiliser **image**.

Syntaxe :
```
r = rotation(O,a);
```

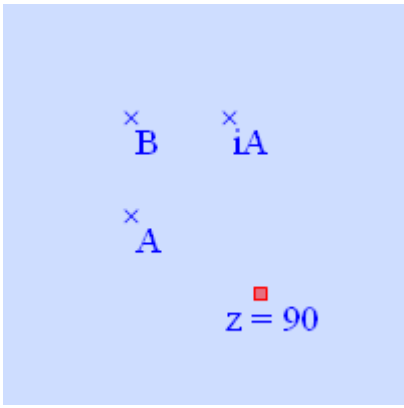
Paramètres :
O: centre de la rotation
a : valeur numérique ou objet numérique (**entier**, **réel**,**var**,...) valeur de l'angle

Exemple 1 :
```
A = point( 2 , 2 );
B = point( 2 , 3.5 );
rB =rotation( B , 90 );
iA =image( rB , A );
```



Place 2 points A et B
puis définit la rotation rB de centre B et d'angle 90°
puis construit l'image iA de A par la rotation rB.

Exemple 2 :
```
A = point( 2 , 2 );
B = point( 2 , 3.5 );
z =reel( 90 , -180 , 180 , 5 );
rB =rotation( B , z );
iA =image( rB , A );
```



Place 2 points A et B
puis définit la variable z, réel valant au départ 90, variant entre -180 et 180 de 5 en 5
puis définit la rotation rB de centre B et d'angle z
puis construit l'image iA de A par la rotation rB.
On peut faire varier la valeur de l'angle en modifiant la valeur de z à la souris (zone rouge).

retour

• reflexion

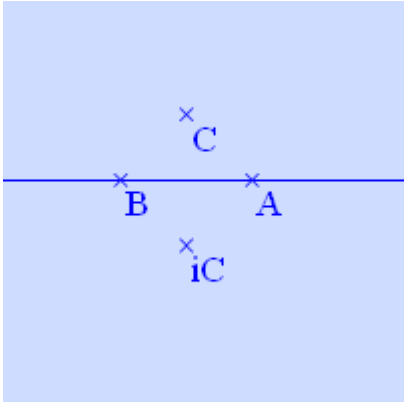
Définit une réflexion par son axe.
Pour construire une image, il faut utiliser **image**.

Syntaxe :
```
r = reflexion(d);
```

Paramètres :
d : droite/segment servant d'axe

Exemple :

```
A = point( 1 , 0 );
B = point( -1 , 0 );
C = point( 0 , 1 );
d =droite( A , B );
sd =reflexion( d );
iC =image( sd , C );
```



Place 3 points A, B et C
puis définit la droite d qui correspond à la droite (AB),
puis définit la réflexion sd d'axe d,
puis construit l'image iA de A par la réflexion sd.

retour

- **segment**

Trace un segment reliant 2 points.

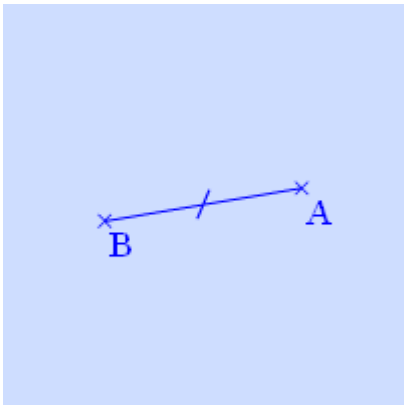
Syntaxe :

```
s = segment(A,B);
```

Paramètres :
A : nom d'une extrémité
B : nom de l'autre extrémité

Exemple :

```
A = point( 2 , 3.5 );
B = point( -1 , 3 );
s =segment( A , B )  { / };
```



Place 2 points nommés A et B puis trace le segment [AB] et le code par un /.

retour

- **segmentLong**

Construit un segment de longueur donnée et le maintient ainsi. L'une ou les 2 extrémités sont créées au besoin.

Syntaxe 1 :

```
s =segmentLong(A,M,x);
```

Paramètres :
A : nom d'une extrémité existante du segment
M : nom de l'autre extrémité qui est créée
x : nombre ou objet numérique (**entier**, **réel**,**var**,...) fixant la longueur de AM

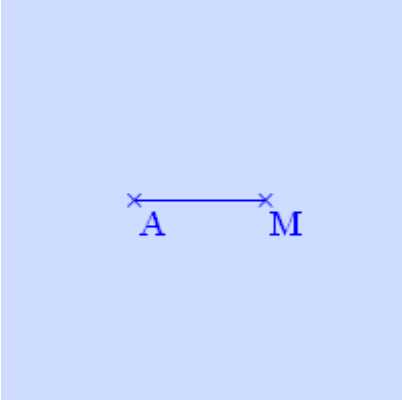
Syntaxe 2 :

```
s =segmentLong(M,N,x);
```

Paramètres :
M : nom d'une extrémité qui est créée
N : nom de l'autre extrémité qui est créée
x : nombre ou objet numérique (**entier**, **réel**,**var**,...) fixant la longueur de AM

Exemple 1 :

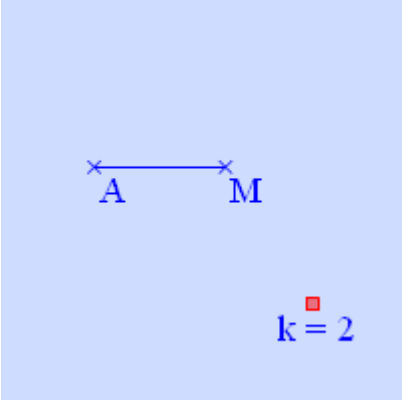
```
A = point( 0 , 0 );
s =segmentlong( A , M , 2 );
```



Place un point A puis construit le segment s d'extrémités A et M (M est créé) pour que AM=2.
En bougeant A ou M le segment mesure constamment 2.

Exemple 2 :

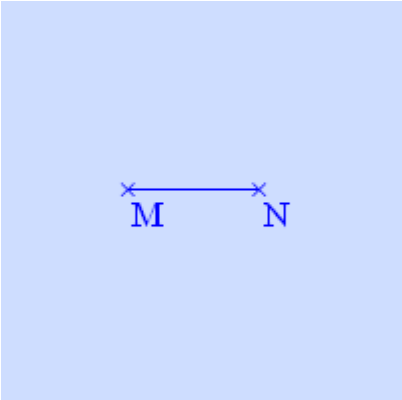
```
A = point( 0 , 0 );
k =reel( 2 , -3 , 3 , 0.1 );
s =segmentlong( A , M , k );
```



Place un point A et définit le réel k de valeur initiale 2 pouvant varier entre -3 et 3 de 0.1 en 0.1 puis construit le segment s d'extrémités A et M (M est créé) pour que AM=k.
En bougeant A ou M le segment mesure constamment k, si on fait varier k le segment change de longueur (M bouge).

Exemple 3 :

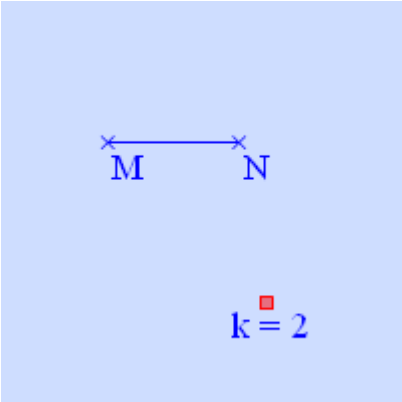
```
s =segmentlong( M , N , 2 );
```



Les 2 points sont créés en même temps que le segment.
Si on bouge le point M ou le point N, la longueur MN fait toujours 2.

Exemple 4 :

```
k =reel( 2 , -3 , 3 , 0.1 );
s =segmentlong( M , N , k );
```



Définit un réel de valeur initiale 2 variant entre -3 et 3 de 0,1 en 0,1,
puis construit le segment s d'extrémités M et N pour que MN=k. M et N sont créés.
Si on bouge le point M ou le point N, la longueur MN fait toujours la valeur de k, si on fait varier k le segment change de longueur (le dernier point sélectionné bouge, N par défaut).

retour

• similitude

Définit une similitude par son centre, son rapport et son angle dans le sens direct du repère. Par défaut l'angle est en degrés.
Pour construire une image, il faut utiliser **image**.

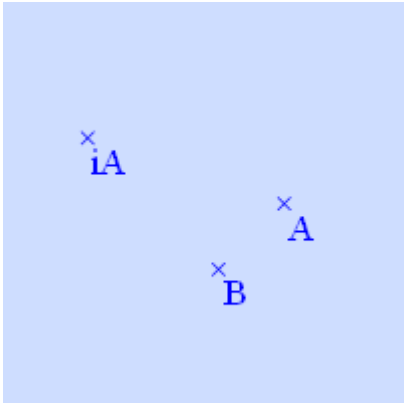
Syntaxe :

```
r = similitude(O,k,a);
```

Paramètres :
O : centre de la similitude
k : nombre ou objet numérique (**entier**, **réel**,**var**,...) valeur du rapport
a : nombre ou objet numérique (**entier**, **réel**,**var**,...) valeur de l'angle

Exemple 1 :

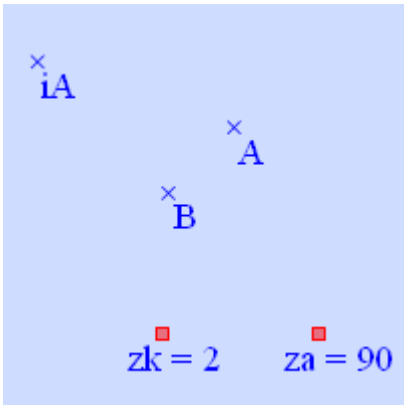
```
A = point( 1 , 1 );
B = point( 0 , 0 );
sB =similitude( B , 2 , 90 );
iA =image( sB , A );
```



Place 2 points A et B,
puis définit la similitude sB de centre B, de rapport 2 et d'angle 90°,
puis construit l'image iA de A par la similitude s.

Exemple 2 :

```
A = point( 1 , 1 );
B = point( 0 , 0 );
zk =reel( 0.5 , 0.5 , 2 , 0.1 );
za =reel( 90 , -180 , 180 , 5 );
sB =similitude( B , zk , za );
iA =image( sB , A );
```



Place 2 points A,B
puis définit la variable zk, valant 0,5 au départ, variant entre 0,5 et 2 de 0,1 en 0,1,
puis définit la variable za, réel valant au départ 90, variant entre -180 à 180 de 5 en 5,
et définit la similitude sB de centre B, de rapport zk, et d'angle za
puis construit l'image iA de A par la similitude s.

retour

• symetrique

Trace le symétrique d'un point par rapport à un point ou à une droite

Syntaxe :

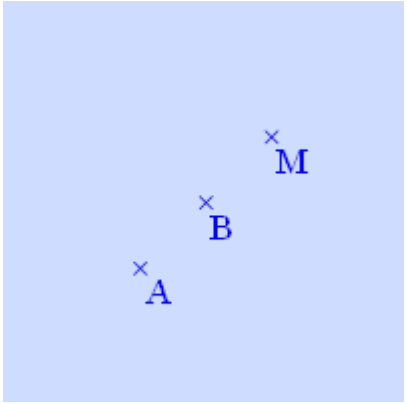
```
M = symetrique(A,d);
```

Paramètres :
A : nom du point
d : axe de symétrie ou centre de symétrie

Exemple 1 :

```
A = point( 0 , 0 );
B = point( 1 , 1 );
M =symetrique( A , B );
```

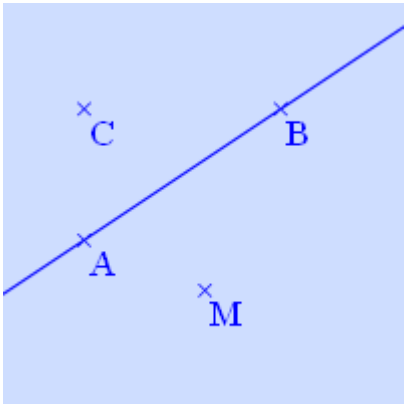
Place 2 points A et B puis construit le symétrique M de A par rapport à B.



Exemple 2 :

```
A = point( 0 , 0 );
B = point( 3 , 2 );
C = point( 0 , 2 );
d =droite( A , B );
M =symetrique( C , d );
```

Place 3 points A, B C
définit la droite (d) passant par A et B
puis construit le symétrique M de C par rapport à (d).



retour

- **symetrie**

Définit une symétrie centrale par son centre ou une symétrie axiale par son axe (voir aussi **reflexion**)

Syntaxe 1 :

```
s = symetrie(A);
```

Paramètres :
A : centre de la symétrie

Syntaxe 2 :

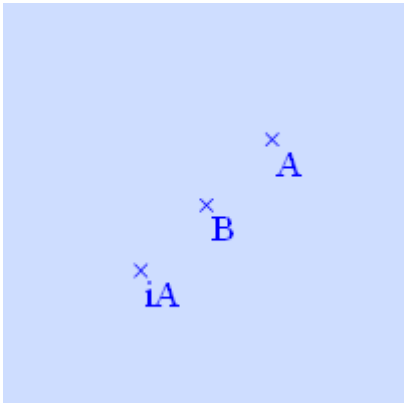
```
M = symetrie(d);
```

Paramètres :
d : axe de la symétrie

Exemple :

```
A = point( 1 , 1 );
B = point( 0 , 0 );
sB =symetrie( B );
iA =image( sB , A );
```

Place 2 points A et B,
définit la symétrie centrale sB de centre B
puis construit l'image iA de A par la symétrie sB.



retour

- **tangente**

Trace la tangente à une courbe de fonction.

Syntaxe 1 :

```
t = tangente(f,k);
```

Paramètres :
f : nom de la courbe de fonction
k : variable ou nombre donnant l'abscisse du point de la courbe dont on veut la tangente.

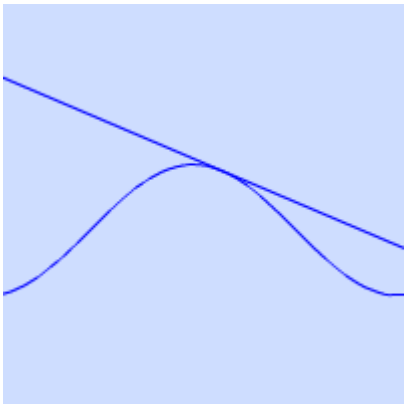
Syntaxe 2 :

```
t = tangente (f,M);
```

Paramètres :
f : nom de la courbe de fonction
M : point dont l'abscisse sera l'abscisse du point de la courbe dont on veut la tangente.

Exemple 1 :

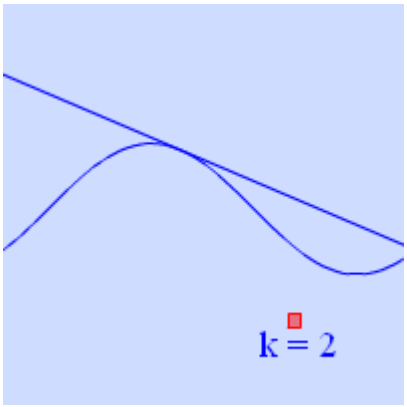
```
@options;
  radians();

@figure;
  f =fonction( sin(x) );
  t =tangente( f , 2 );
```



Trace dans le repère courant la courbe représentative de la fonction f : [y=sin(x)] puis sa tangente t au point d'abscisse 2.

Exemple 2 :

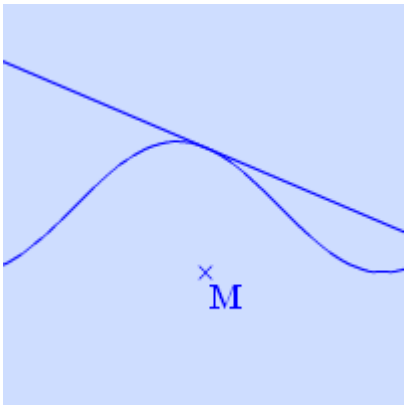
```
@options;
  radians();

@figure;
  f =fonction( sin(x) );
  k =reel( 2 , -5 , 5 , 0.2 );
  t =tangente( f , k );
```



Trace dans le repère courant la courbe représentative de la fonction f : [y=sin(x)] et définit le réel k de valeur initiale 2 pouvant varier entre -5 et 5 de 0.2 en 0.2.
puis trace la tangente t à la courbe f au point d'abscisse k.

Exemple 3 :

```
@options;
  radians();

@figure;
  f =fonction( sin(x) );
  M = point( 2 , -1 );
  t =tangente( f , M );
```



Trace dans le repère courant la courbe représentative de la fonction f : [y=sinx(x)]
puis place un point M d'abscisse 2 (et d'ordonnée -1)
et trace la tangente t à la courbe f au point dont l'abscisse est celle de M.

Quand M est déplacé, la tangente suit son abscisse.
On peut plutôt utiliser un PointSur la courbe.

retour

- **texte**

Affiche une zone texte positionnée librement dans le repère actuel ou liée à un point.

Syntaxe 1 :

```
t = texte(x,y,"caractères");
```

Paramètres :
x : position en abscisse
y : position en ordonnée
"caractères" : texte à afficher

Syntaxe 2 :

```
t = texte(M,"caractères");
```

Paramètres :
M : point déterminant la position
"caractères" : texte à afficher

"caractères" permet, outre du texte brut, d'afficher :

- **la valeur d'une variable :**
  il suffit de mettre dans le texte entre guillemets le nom de la variable encadrée par des caractères $:
  t=texte(M,"La variable v vaut v.");

- **le résultat d'une commande de la zone analyse :**
  il suffit de mettre dans le texte entre guillemets la commande de la zone analyse encadrée par des caractères # :
  t=texte(M,"Les droites (d) et (d') sont #position(d,d')=#");

- **des symboles particuliers :**
  il suffit de mettre dans le texte entre guillemets l'abréviation du symbole, encadrée par des caractères £ :
  les lettres grecques :
  t=texte(M,"L'angle £alpha£ est compris entre 0° et 90°");

  alpha - beta - chi - delta - epsilon - phi - gamma - eta - iota - phi - kappa - lambda - mu - nu
  omicron - pi - theta - rho - sigma - tau - upsilon - sampi - omega - xi - psi – dzeta

  α - β - χ - δ - ε - ϕ - γ - η - ι - φ - κ - λ - μ - ν
  ο - π - θ - ρ - σ - τ - υ - ϖ - ω - ξ - ψ - ζ

  Les versions majuscules s'obtiennent en mettant la première lettre en Majuscule : £alpha£ donne £Alpha£ en majuscule :

  Α - Β - Χ - Δ - Ε - Φ - Γ - Η - Ι - ϑ - Κ - Λ - Μ - Ν
  Ο - Π - Θ - Ρ - Σ - Τ - Υ - ς - Ω - Ξ - Ψ - Ζ

  des symboles mathématiques :
  t=texte(M,"(MA)£perp£(MB) et AB£environ£3"); donnera

  (MA)⊥(MB) et AB≈3

  Voici la liste des codes utilisables pour ces symboles :

petitf	f
moins	-
petitebarre	–
grandebarre	—
prime	′
seconde	″
puceronde	•
grandC	$\mathbb{C}$
euler	$\mathscr{E}$
petitg	$\mathscr{g}$
petith	$\mathscr{h}$
Ironde	$\mathscr{I}$
Lronde	$\mathscr{L}$
Cronde	$\mathscr{l}$
grandN	$\mathbb{N}$
Pronde	$\wp$
grandQ	$\mathbb{Q}$
Rronde	$\Re$
grandR	$\mathbb{R}$
grandZ	$\mathbb{Z}$
Eronde	$\mathscr{E}$
Fronde	$\mathscr{F}$
Nronde	$\aleph$
flecheG	←
flecheH	↑
flecheD	→
flecheB	↓
flecheDG	↔
flecheGD	↔

flecheHB	↕
flecheBH	↕
croissant	↗
descroissant	↘
alaligneadroite	↳
alaligneagauche	↲
doubleflecheG	⇐
doubleflecheD	⇒
doubleflecheDG	⇔
doubleflecheGD	⇔
flecheGbarre	⇤
flecheDbarre	⇥
flecheGcreuse	⇦
flecheHcreuse	⇧
flecheDcreuse	⇨
flecheBcreuse	⇩
qqsoit	∀
pourtout	∀
quelquesoit	∀
differentielpartiel	∂
ilexiste	∃
vide	∅
nabla	∇
appartienta	∈
nappartientpasa	∉
contient	∋
petitcontient	∋

grandproduit	∏
grandcoproduit	∐
grandesomme	Σ
petitebarrefine	–
moinsouplus	∓
antislash	\
asterisque	∗
racine	√
proportionnela	∝
infini	∞
angle	∠
anglespherique	∢
divise	|
nedivisepas	∤
parallelea	||
etlogique	∧
oulogique	∨
inter	∩
intersection	∩
union	∪
integrale	∫
doubleintegrale	∫∫
tripleintegrale	∫∫∫
integralecurviligne	∫ ou avec o
integralesurfacique	∯
integralevolumique	∰
egaleasymptotiquea	≃
environdroit	≃

environegala	≈
environ	≈
egalpardefinition	≝
differentde	≠
identiquea	≡
inferieura	≤
superieura	≥
inclusdans	⊂
contient2	⊃
nestpasinclusdans	⊄
sommedirecte	⊕
differencedirecte	⊖
produittensoriel	⊗
divisiondirecte	∅
produitdirect	⊙
top	⊤
perpendiculairea	⊥
perp	⊥
antecedentde	⊷
imagede	⊸
angledroitarc	⊾
point	•

- **des formules mathématiques :**
  Une syntaxe est prévue pour mettre en forme un texte contenant des formules mathématiques.
  Elle se base sur le caractère **£** comme marqueur spécifique, suivi d'**une lettre** indiquant la fonction appliquée à un texte qui suit **entre parenthèses**.

  Voici la liste des fonctions :

lettre	fonction	exemple	affichage
a	angle	`£a(ABC)`	$\widehat{ABC}$
c	crochets	`£c(AB)`	[AB]
e	mise en exposant	`£e(3,2)`	3^2
f	fraction	`£f(1,2)`	$\frac{1}{2}$
i	mise en indice	`£i(M,1)`	M_1
p	parenthèses	`£p(AB)`	(AB)
r	racine carrée	`£r(2)`	$\sqrt{2}$
v	vecteur	`£v(AB)`	$\overrightarrow{AB}$

  On peut utiliser les autres possibilités de la syntaxe de texte : variable entre $, résultat d'analyse entre # et symboles entre £.

  Exemple :

```
var x = 2;
t1=texte(0,0,"£f(£pi£,2) et £r($x$)");
```

  affiche : $\frac{\pi}{2}$ et $\sqrt{2}$

- **des embellissements supplémentaires :**
  une partie des codages HTML de mise en forme est disponible pour personnaliser un texte dans TracenPoche en dehors des styles prédéfinis. Éviter les guillemets " comme séparateurs en utilisant l'apostrophe '.

  Exemple :
  t=texte(M,"< font color='#AA8866' size='12' face='Arial'>mon texte< /font>");
  affiche mon texte en beige foncé, taille 12 avec la police Arial :

mon texte

Exemple 1 :

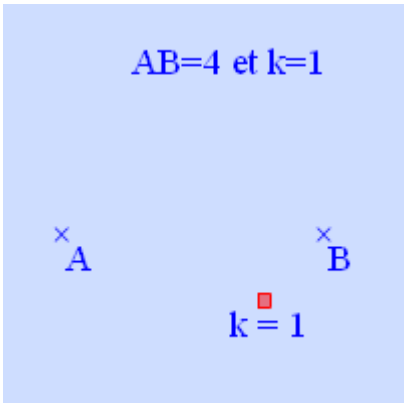
```
t =texte( 0 , 0 , "coucou");
```

Affiche le texte coucou au point (0,0) du repère.

Exemple 2 :

```
A = point( -2 , -1 );
B = point( 2 , -1 );
k =reel( 1 , -5 , 5 , 0.1 );
var z =AB { 4 };
t =texte( -1 , 2 , "AB=$z$ et
k=$k$") {dec1};
```

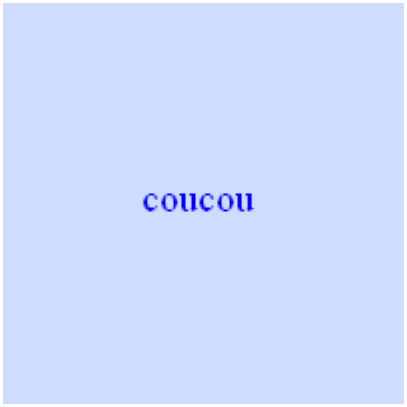
Place 2 points A et B,
définit le reel k et la distance z de A à B,
puis affiche au point (-2,2) le texte : AB=4 et k=1.
L'option {dec1} précise que l'affichage se fait avec un chiffre après la virgule si besoin est.
Si on bouge A, B ou si on fait varier k, le texte se met à jour.



Exemple 3 :

```
A = point( 0 , 1 )  { i };
t =texte( A , "coucou");
```

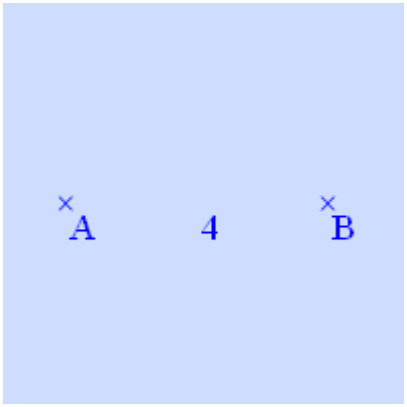
Place le point A et le cache,
puis affiche le texte coucou à la place du point A.



Exemple 4 :

```
A = point( -2 , -1 );
B = point( 2 , -1 );
M =milieu( A , B )  { i };
var d =AB { 4 };
t =texte( M , "$d$")  { dec2 };
```

Place 2 points A et B, construit et cache le milieu M du segment [AB],
définit la distance d de A à B,
puis affiche au point M la distance d.



retour

• translation

Définit une translation par son vecteur ou par un point et son image.
Pour construire une image, il faut utiliser **image**.

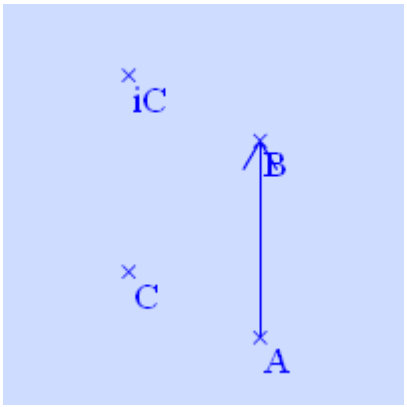
Syntaxe 1 :
```
t = translation(v);
```

Paramètres :
v : vecteur

Syntaxe 2 :
```
t = translation( A , B );
```

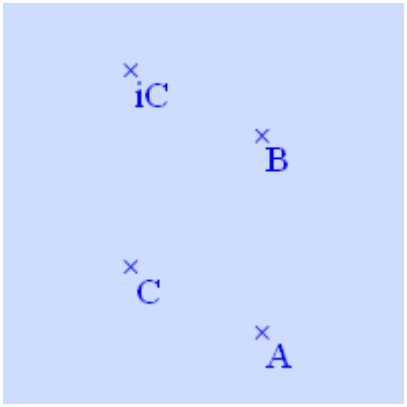
Paramètres :
A : point origine
B : point image
C'est la translation qui transforme A en B

Exemple 1 :
```
A = point( 1 , 0 );
B = point( 1 , 3 );
C = point( -1 , 1 );
v =vecteur( A , B );
tv =translation( v );
iC =image( tv , C );
```



Place 3 points A,B,C et définit le vecteur v de A vers B,
puis définit la translation tv de vecteur v (qui transforme A en B),
puis construit l'image iC de C par la translation tv.

Exemple 2 :
```
A = point( 1 , 0 );
B = point( 1 , 3 );
C = point( -1 , 1 );
t =translation( A , B );
iC =image( t , C );
```



Place 3 points A,B,C,
puis définit la translation t qui transforme A en B,
puis construit l'image iC de C par la translation t.

retour

• variable

Définit une variable et évalue sa valeur.
Le résultat de l'évaluation apparaît entre accolades après l'expression et avant le point virgule.

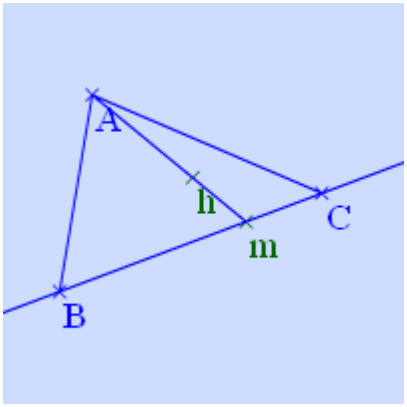
Syntaxe :
```
var x = expression;
```

Paramètres :
x : nom de la variable
expression : une expression algébrique quelconque

le résultat de l'évaluation apparaît entre accolades après l'expression et avant le point virgule

Exemple :

```
@options;
  radians();

@figure;
  A = point( -1.5 , 2 );
  B = point( -2 , -1 );
  C = point( 2 , 0.5 );
  sAB =segment( A , B );
  dBC =droite( B , C )  { sansnom };
  sCA =segment( C , A );
  var x =AB/BC { 0.711933504667742 };
  m =pointsur( B , C , x )
{ vertfonce };
  sAm =segment( A , m );
  var y =sin(x) { 0.653298847914552 }
;
  h =pointsur( A , m , y )
{ vertfonce };
```



Opérateur	Emploi
+	addition
-	soustraction
*	multiplication
/	division
^	puissance

Fonction Calc.	Emploi
carre	`var x = carre(2);`
racine	`var x = racine(2);` racine carrée
abs	`var x = abs(-2);` valeur absolue ou distance à zéro
ceil	`var x = ceil(1.5);` valeur arrondie à l'unité
ent	`var x = ent(1.5);` partie entière
ln	`var x = ln(2);` logarithme népérien
exp	`var x = exp(2);` exponentielle
cos	`var x = cos(30);` cosinus
sin	`var x = sin(30);` sinus
tan	`var x = tan(30);` tangente
acos	`var x = acos(0.5);` arccosinus
asin	`var x = asin(0.5);` arcsinus
atan	`var x = atan(0.5);` arctangente

Fonction TeP	Emploi
aire	`var x = aire(ABC);`
angle	`var x = angle(ABC);` angle saillant non orienté (entre 0° et 180° ou 0 rad et pi rad) `var x = ABC;` angle orienté saillant (entre -180° et 180° ou -pi rad et pi rad)
anglev	`var x = anglev(ABC);` angle non orienté entre 0° et 360° (ou 0°rad et 2pi rad)
arc	`var x = arc(ABC);`
distance	`var x = AB;`
perimetre	`var x = perimetre(ABC);`
abscisse	`var x = abscisse(A);`

ordonnee	`var x = ordonnee(A);`
random	`var x = random();` renvoie un réel de l'intervalle [0,1[(0 inclus, 1 exclus) `var x = random(3,5);` renvoie un ENTIER de l'intervalle [3,5] : 3,4 ou 5 `var x = random(3);` équivalent à random(0,3)
max	`var x = max(y,z,...);` renvoie le max des nombres de la liste (y,z,...)
min	`var x = min(y,z,...);` renvoie le min des nombres de la liste (y,z,...)
absc	`var x= absc(A);` où A est un pointsur, renvoie l'abscisse de A sur la droite

retour

• varsi

Définit une variable dont la valeur (numérique ou texte) dépend d'une condition.
Le valeur de cette variable apparaît entre accolades après l'expression et avant le point virgule.

Syntaxe :

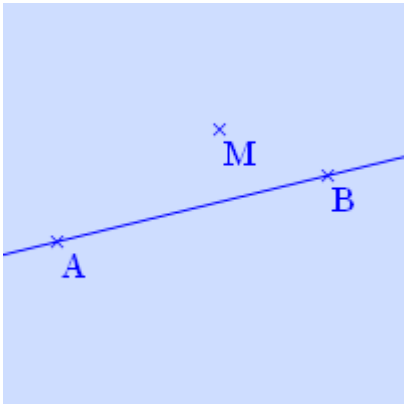
```
varsi  z  =  [ condition , EA1 ou
"TEXTE1" , EA2 ou "TEXTE2"  ]
```

Paramètres :
condition : sous la forme EA opérateur EA, ou point sur point, ou point appartient droite.
EA : expression algébrique

Exemple 1 :

```
  A = point( -1.8 , 1.7 );
  B = point( 2.7 , 2.8 );
  dAB = droite( A , B );
  M = point( 0.9 , 3.5 );
  varsi z =[MappartientAB_5%,1,0];
```

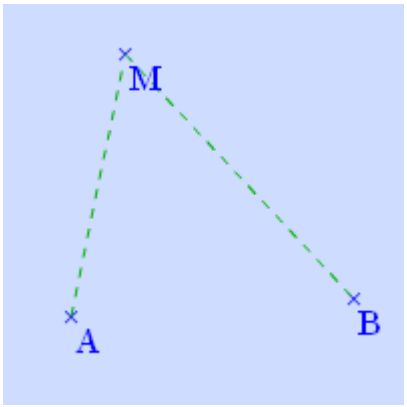
La variable z vaut 1 si M appartient à la droite (AB) à 0,05 près, elle vaut 0 sinon.



Exemple 2 :

```
  A = point( -3 , 0 );
  B = point( 1.7 , 0.3 );
  M = point( -0.8 , 4.4 );
  varsi z =[AM > MB,"rouge-3","vert-
7"];
  sAM = segment( A , M )  {$z$};
  sMB = segment( M , B )  {$z$};
```

Les segments [AM] et [MB] sont en rouge et épais si AM>BM, sinon ils sont verts et en pointillés.



retour

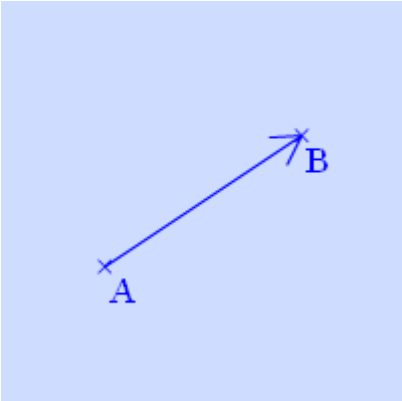
• vecteur

Trace le représentant d'un vecteur figuré par une flèche reliant 2 points.

Syntaxe 1 :
```
v = vecteur (A,B);
```

Paramètres :
A : nom de l'origine du vecteur
B : nom de l'extrémité du vecteur (flèche)

Exemple :
```
A = point( -1 , -1 );
B = point( 2 , 1 );
v = vecteur( A , B );
```



Place 2 points nommés A et B puis trace le vecteur $\overrightarrow{AB}$.

Syntaxe 2 :
```
vecteur v = combinaison;
```

Paramètres :
v : nom du vecteur
combinaison : formule exprimant une combinaison linéaire d'autres vecteurs

Exemple :
```
A = point( 0.17 , 2 );
B = point( -4.77 , -2.7 );
vAB = vecteur( A , B );
u = vecteurcoord( 4 , 0 );
vecteur w = 2vAB + u ;
```

Définit un vecteur $\vec{w}$ égal à $2\overrightarrow{AB} + \vec{u}$. Ce vecteur n'est pas tracé.

retour

- **vecteurCoord**

Définit un vecteur par ses coordonnées dans le repère actuel. Aucun représentant n'est dessiné.

Syntaxe :
```
v = vecteur (x,y);
```

Paramètres :
x : abscisse (un nombre)
y : ordonnées (un nombre)

Exemple :
```
v =vecteurcoord( 2 , 1 );
```

retour

3.3 Les options d'une figure

Ces commandes sont à utiliser dans la section **@options**.

A
aimante

• aimante

Rend la grille du repère actuel aimantée : les déplacements de la souris ne s'opèrent que sur les noeuds de cette grille.

Syntaxe :

```
aimante();
```

Paramètres :

retour

• chgt_etat_bloc

Permet le changement d'état d'un ou plusieurs objets par l'appui simultané sur le bouton gauche de la souris et sur une touche du clavier. Au deuxième clic+touche, le ou les objets reprennent leur état initial défini dans la section **@figure**.

Syntaxe :

```
chgt_etat_bloc ("touche", {option_1,
option_2, ...}, objet_1, objet_2,
..., {option_i, ...}, objet_n, ... )
;
```

Paramètres :

- "touche" : la touche du clavier qui permet de changer l'état du ou des objets (à écrire entre guillemets, minuscule ou majuscule), sauf les touches P et S,
- {option_1,option_2,...} : liste des options caractérisant l'aspect du ou des objets dont le ou les noms vont suivre,
- objet_1,objet_2,... : liste des objets qui prendront l'aspect défini par les options précédentes,
- {option_i,...} : nouvelle liste d'options définissant un état.
- objet_n,... : nouvelle liste d'objets qui prendront l'aspect défini précédemment.

On peut mettre autant de couples listes d'options/objets que l'on souhaite.

Les options que l'on peut modifier dépendent des objets auxquels elles s'appliquent :
- pour les points et lignes : couleu par son nom ou son codage héxadécilmal, visibilité (**i** ou **v**) , style (**1,2,3,rond1,rond2,rond3,7,8,9** ; pour les lignes) , présence du nom (**avecnom,sansnom**), la trace (**trace**).
- pour les cercles : couleur, visibilité (**i** ou **v**) , style (**1,2,3,7,8,9**).
- pour les polygones : couleur, visibilité (**i** ou **v**) , style (**1,2,3**) , **pleinxx**

On peut créer plusieurs changements d'état **chgt_etat_bloc** en utilisant différentes touches clavier, sauf P (mode Pas à Pas) et S (actualisation du Script). L'appel à un changement d'état par une touche annule l'appel à un éventuel changement d'état précédent : la figure utilisée pour appliquer le changement d'état est bien la figure telle que définie par sa construction dans le script **@figure**.

retour

• degres

Fixe l'unité de mesure des angles au degré. C'est l'unité par défaut.
On peut aussi utiliser le radian.

Syntaxe : *Paramètres :*

```
degres();
```

retour

• grille

Rend visible la grille du repère actuel (que le repère soit visible ou pas et quelles que soient les étiquettes de graduation).

Syntaxe : *Paramètres :*

```
grille();
```

retour

• radians

Fixe l'unité de mesure des angles au radian.
Par défaut, l'unité employée est le degré.

Syntaxe : *Paramètres :*

```
radians();
```

retour

• repere

Définit l'étendue et les graduations du repère du plan utilisé pour construire. Les axes de coordonnées sont perpendiculaires (axe des abscisses horizontal, axe des ordonnées vertical)
On peut montrer la grille du repère et l'aimanter.

Syntaxe :

```
repere(x_min, x_max, y_min,y_max,
gradu_x,gradu_y) {options};
```

Paramètres :
x_min : minimum de l'axe des abscisses
x_max : maximum de l'axe des abscisses
y_min : minimum de l'axe des ordonnées
y_max : maximum de l'axe des ordonnées
gradu_x : pas de marquage des graduations en abscisses
gradu_y : pas de marquage des graduations en ordonnées

options :
- style de la police : 0=normal, 1=italique, 2=gras, 3=grasitalique
- taille de la police petit=taille 10, moyen=taille 12, grand=taille 14
- couleur de la police : rouge, vert ... (voir options{})
- étiquettes des abscisses : num1=étiquette à chaque graduation, num2=étiquette toutes les 2 graduations, num3, num4, num5, num10, num20
- visibilité du repère : i=repère invisible, le repère est visible si ce paramètre est absent

retour

- **repereOrtho**

Définit l'étendue et les graduations du repère du plan utilisé pour construire en le rendant orthonormé et non simplement orthogonal.
On peut montrer la grille du repère et l'aimanter.

Syntaxe 1 :

```
repereortho(x_origine, y_origine,
unite, gradu_x,gradu_y) {options};
```

Paramètres :
x_origine : position horizontale en pixels de l'origine du repère à l'écran
y_origine: position verticale en pixels de l'origine du repère à l'écran
unite : nombre de pixel définissant l'unité du repère
gradu_x : pas de marquage des graduations en abscisses
gradu_y : pas de marquage des graduations en ordonnées

options :
- style de la police : 0=normal, 1=italique, 2=gras, 3=grasitalique
- taille de la police petit=taille 10, moyen=taille 12, grand=taille 14
- couleur de la police : rouge, vert ... (voir options{})
- étiquettes des abscisses : num1=étiquette à chaque graduation, num2=étiquette toutes les 2 graduations, num3, num4, num5, num10, num20
- visibilité du repère : i=repère invisible, le repère est visible si ce paramètre est absent

Syntaxe 2 :

```
repereortho(min_x,max_x,min_y,max_y)
{options};
```

Paramètres :

Calcule les valeurs à utiliser dans la syntaxe précédente pour que le repère affiche au moins les graduations de min_x à max_x en abscisses et de min_y à max_y en ordonnées (cadrage)

retour

- **trame**

Rend visible la grille du repère actuel (que le repère soit visible ou pas). La grille peut être aimantée.
Commande analogue à l'option grille(), un réseau de ligne apparaît plutôt qu'un réseau de point.
Cette commande permet d'obtenir une exportation vers OOo avec du papier millimétré.

Syntaxe :

```
trame();
```

Paramètres :

retour

4. Les compléments de TeP

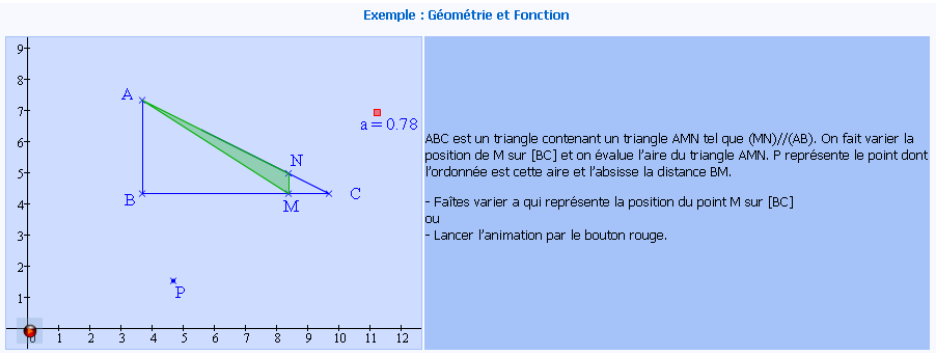
4.1 TepWeb

4.1.1 Présentation

TepWeb est une version allégée de TracenPoche permettant l'insertion d'une figure dynamique dans une page web.

Le bas de la page d'accueil du site (www.tracenpoche.net) en montre un exemple d'utilisation.

Un exemple de fonctionnalités



Outre la légèreté de l'interface qui permet un chargement plus rapide des pages, l'autre intérêt de TepWeb est que l'interface ne nécessite pas d'autre installation que le seul lecteur Flash © Macromedia qui se fait de manière invisible la première fois (à priori si on navigue unt peu sur le web, on a déjà ce lecteur installé sans le savoir).

On peut télécharger l'archive TepWeb.zip depuis la rubrique **`téléchargement`** du site : l'archive contient les fichiers tepwebxy.swf, un fichier HTML (exemple.htm) et un fichier script texte (script.txt) plus un base.tep et macro.tep.

Par défaut TepWeb est en 300 pixels par 300 pixels (1 pixel = 1 point affiché par le moniteur).

Mais plusieurs TepWeb sont également disponibles dans l'archive :
- tepweb150150
- tepweb200200
- tepweb400300
- tepweb400400
- tepweb450450
- tepweb500500
- tepweb600600
- tepweb900300
- tepweb600500

4.1.2 Utilisation de TepWeb

Le code HTML permettant l'insertion d'une figure script1.txt dans une page web ressemble à celui-ci :

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#v
  ersion=7,0,0,0" width="300" height="300" align="middle">
    <param name="allowScriptAccess" value="sameDomain" />
    <param name="quality" value="high" />
    <param name="bgcolor" value="#ffffff" />
    <param name="SRC" value="tepweb.swf?script=script1.txt">
    <embed src="tepweb.swf?script=script1.txt" width="300" height="300"
      align="middle" quality="high" bgcolor="#ffffff" swLiveConnect=true
      id="script1" allowScriptAccess="sameDomain" type="application/x-shockwave-
      flash" pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

On remarque qu'il y a une double déclaration de l'objet TepWeb ("objet" est conforme au standard HTML 4 supporté par les navigateurs les plus courants, "embed" n'est pas dans le vocabulaire HTML mais assure la compatibilité avec d'anciens navigateurs), donc on retrouve 2 fois les paramètres `width=""` et `height=""` ainsi que la valeur `tepweb.swf?script=script1.txt`.

Par conséquent, si on veut utiliser tepweb500500 il faut penser à modifier le code comme ceci :

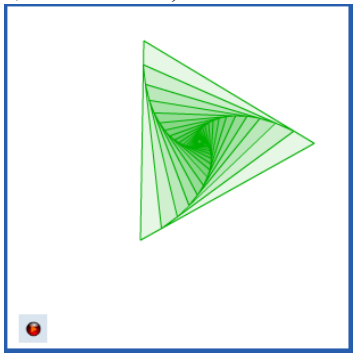
```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#v
  ersion=7,0,0,0" width="500" height="500" align="middle">
    <param name="allowScriptAccess" value="sameDomain" />
    <param name="quality" value="high" />
    <param name="bgcolor" value="#ffffff" />
    <param name="SRC" value="tepweb500500.swf?script=script1.txt">
    <embed src="tepweb500500.swf?script=script1.txt" width="500" height="500"
      align="middle" quality="high" bgcolor="#ffffff" swLiveConnect=true
      id="script1" allowScriptAccess="sameDomain" type="application/x-shockwave-
      flash" pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Les fichiers TeP chargés par l'interface TepWeb, qui doivent impérativement se trouver dans le même répertoire que la page web, doivent avoir la structure suivante :

```
@options
      options de la figure

@figure
      le script de la figure

@config
      couleurfonddessin=0xFEFDDE;
      boutons=animation;
```

} cette section est facultative

On ne peut donc changer que la couleur du fond à l'aide du paramètre **`couleurfonddessin`** et éventuellement afficher le bouton animation si dans le script une variable (**`entier`** ou **`reel`**) est animée (**`anime`**, **`anime1`**, **`oscille`**, **`oscille1`**, **`oscille2`**).



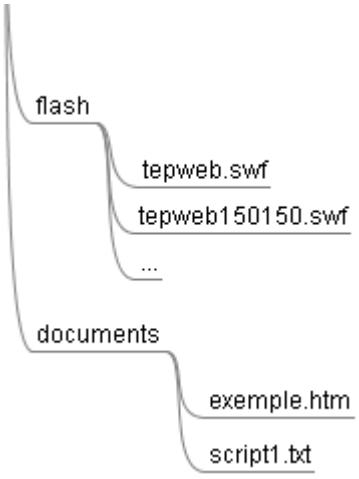
ici fond blanc et bouton animation

Ces 2 configurations sont donc facultatives.

L'applet peut être partagée par plusieurs dossiers, sans avoir à la copier dans chaque dossier.
Il suffit de mettre le chemin relatif d'accès à tepweb.swf par rapport au dossier contenant la page HTML :

```
...
<param name="SRC" value="../flash/tepweb.swf?script=script1.txt">
<embed src="../flash/tepweb.swf?script=script1.txt" width= ...
...
```

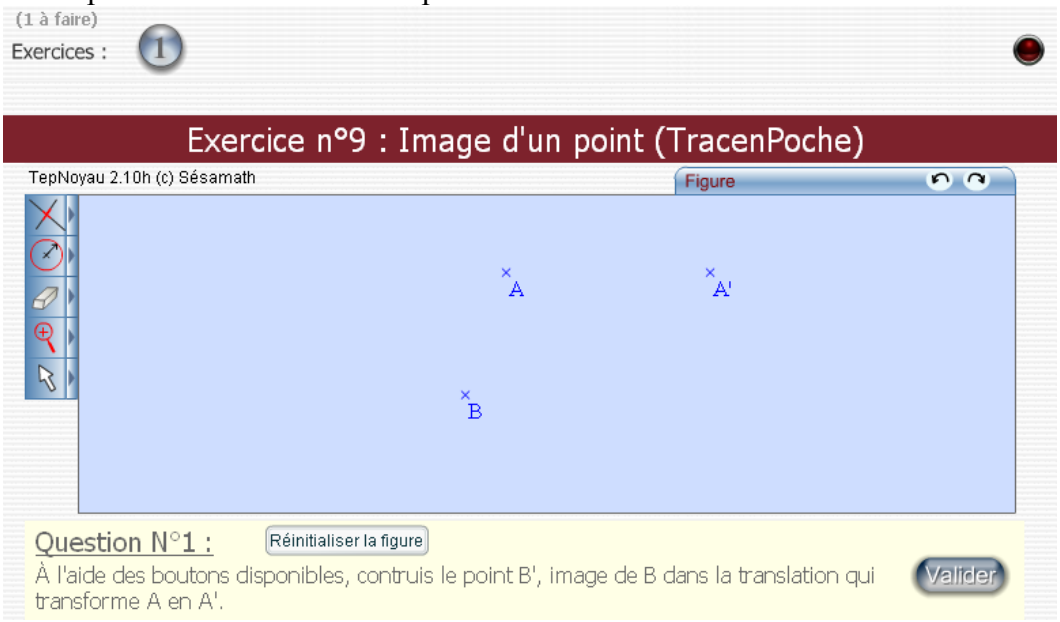
Dans cet exemple, la page HTML est dans un sous-dossier *documents* et l'applet est dans un autre sous dossier de même niveau nommé *flash* :



Ce qui importe ici, c'est que le fichier script1.txt soit dans le même dossier que la page dans laquelle est insérée l'applet TepWeb qui appelle ce script.

4.2 TepNoyau

TepNoyau est une version adaptée de TracenPoche permettant l'insertion de TracenPoche dans une animation comme pour les exercices Mathenpoche.



TeP dans MeP

TepNoyau est réservé pour l'instant aux développeurs Mathenpoche. Elle n'est pas distribuable ou copiable sans autorisation.

TepNoyau est constitué d'une seule animation tepnoyau.swf : elle est externe et doit donc être chargée à partir du fichier de développement écrit Action Script/Flash. Cela permet un développement séparé de l'animation finale et de TepNoyau.

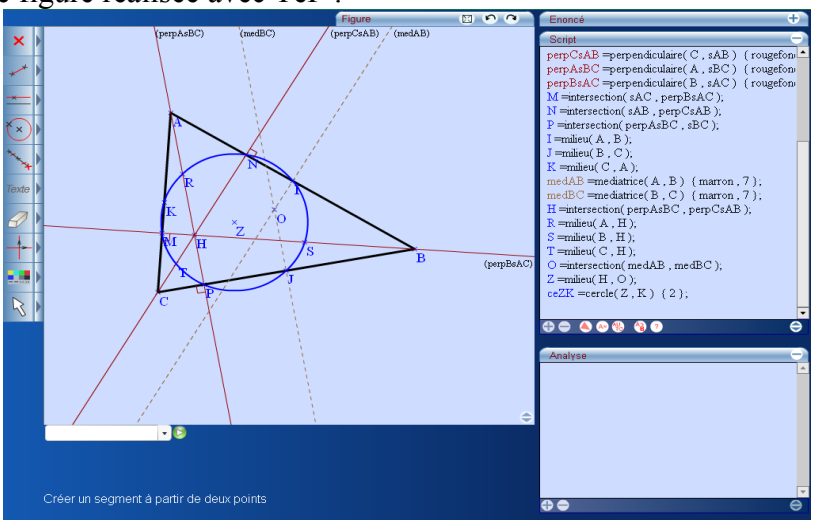
Il permet :
- l'insertion d'une figure dynamique dans un fichier source (exercice Mathenpoche).
- le paramétrage complet des zones, des boutons et des commandes disponibles
- le dialogue entre la figure et l'animation Flash produite : ce qui permet de questionner la figure et de créer des exercices de géométrie avec contrôle des actions/constructions.

4.3 OOoTeP

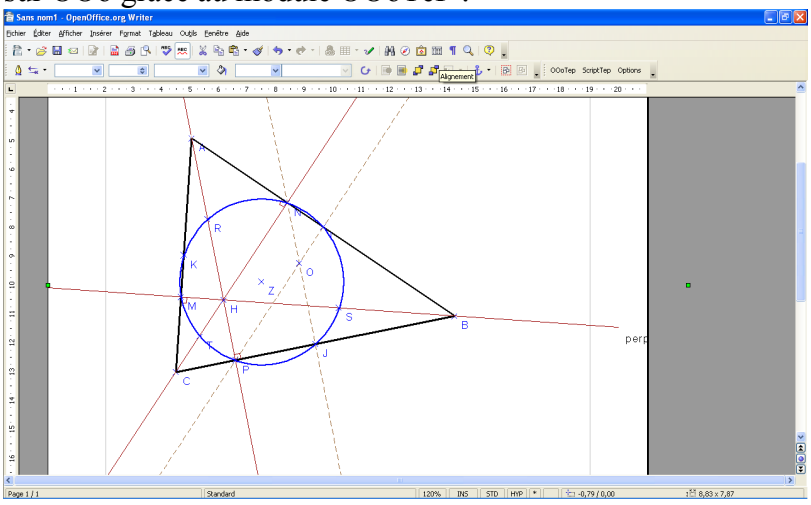
4.3.1 Présentation

OOoTeP est un module écrit pour OpenOffice.org 2.0 pour insérer en vectoriel des images-figures produites par TeP.

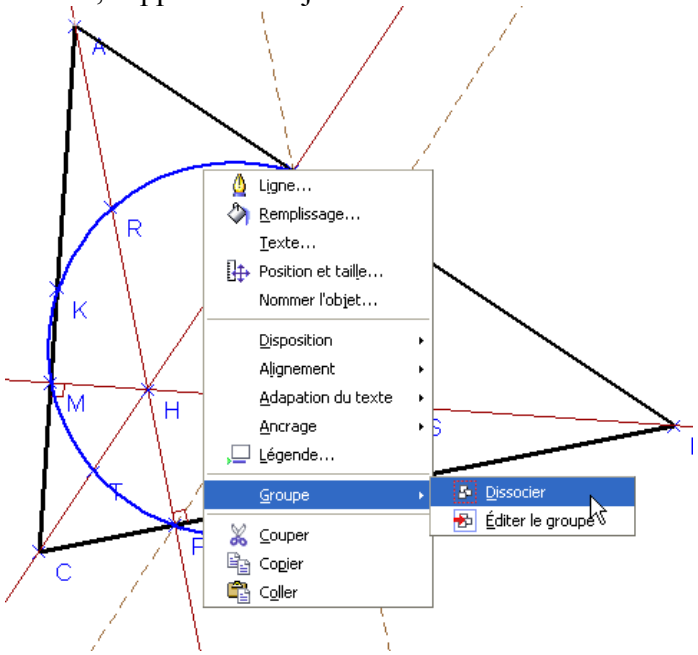
Par exemple, une telle figure réalisée avec TeP :



pourra être récupérée sur OOo grâce au module OOoTeP :



Un des avantages de OOoTeP est que l'image obtenue est une image vectorielle : on peut en changer les dimensions sans en altérer le rendu (pas d'effet d'escalier ou de damier) et rien n'empêche par la suite de dissocier l'image afin de modifier, supprimer ou ajouter des éléments à sa convenance :



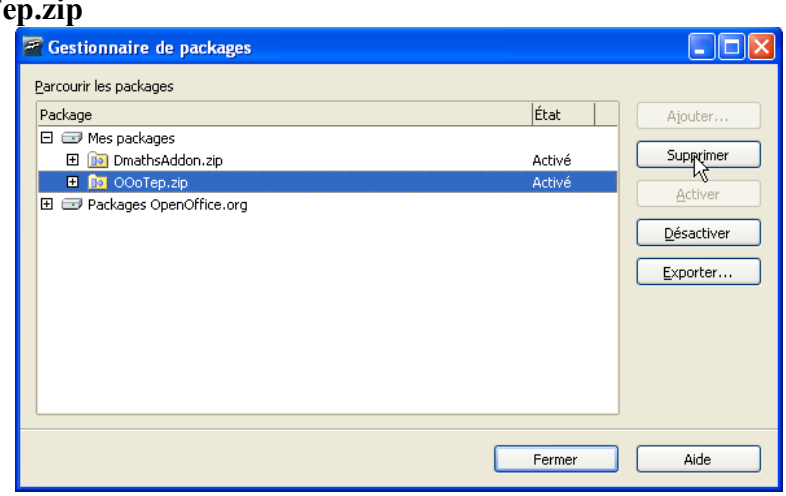
Enfin, plus fort encore, la figure importée dans OOo pourra ensuite être à nouveau modifiée dans TeP !

4.3.2 Installation du module.

On peut télécharger le module dans la rubrique **`téléchargement`** du site.
Il se présente sous la forme d'une archive nommée oootep.zip. Cette archive n'a pas à être décompressée pour installer le module OOoTeP dans OOo.

*Si un module OOoTeP est **déjà** installé, il faut commencer par le désinstaller :*

1. Cliquer sur **Outils / Gestionnaire de packages ...**
2. Cliquer sur le + devant **Mes packages**
3. Sélectionner **OooTep.zip**

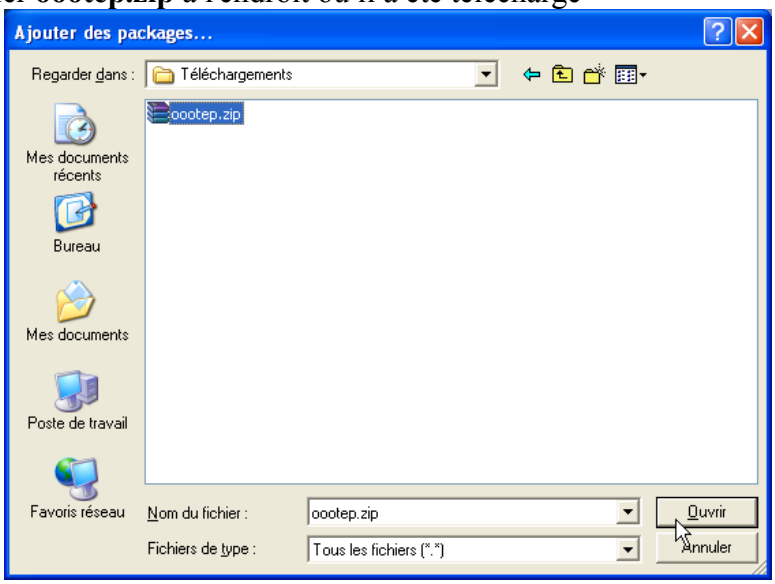


4. Cliquer sur **Supprimer**
5. Cliquer sur **Fermer**

`OOoTep est désinstallé !`

Installation du module OOoTep :

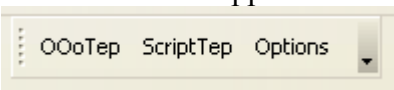
1. Cliquer sur **Outils / Gestionnaire de packages ...**
2. Sélectionner **Mes packages**
3. Cliquer sur **Ajouter**
4. Sélectionner le fichier **oootep.zip** à l'endroit où il a été téléchargé



5. Cliquer sur Ouvrir
6. Le module oootep.zip apparaît dans **Mes packages**
7. Cliquer sur **Fermer**

```
OOoTeP est installé !
```

8. **Quitter** OOo et le **relancer**
9. Une nouvelle barre d'outils contenant 3 boutons est apparue



4.3.3 Utilisation du module

a) Création d'une figure avec OOoTeP

Lancer TracenPoche.

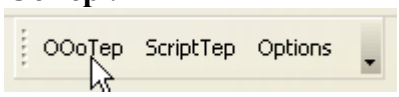
Créer par exemple la figure suivante:

```
@options;
@figure;
  A = point( -0.53 , 0.83 )  { noir , car+3 , italique };
  B = point( -0.23 , -5.33 );
  sAB = segment( A , B );
  C = point( 4.6 , 0.03 );
  sBC = segment( B , C )  { noir , 2 };
  sAC = segment( A , C )  { 3 };
  ceAB = cercle( A , B )  { rougefonce , 9 };
```

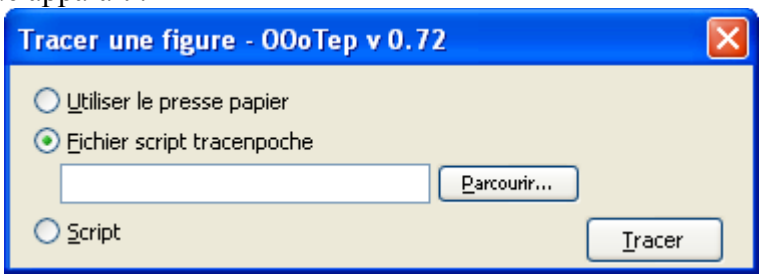
Dans TeP, la dernière rangée de boutons comporte un bouton OOo  qui permet de produire le fichier texte pour OOoTeP : c'est un script TeP avec quelques informations supplémentaires pour positionner correctement les objets.
On peut donc copier ce script texte OOoTeP pour le coller, et éventuellement le sauvegarder dans un fichier texte.

NB : dans la version en ligne du logiciel TracenPoche, le bouton  permet de générer le fichier texte.

Dans OOo, cliquer sur le bouton **OOoTep** :



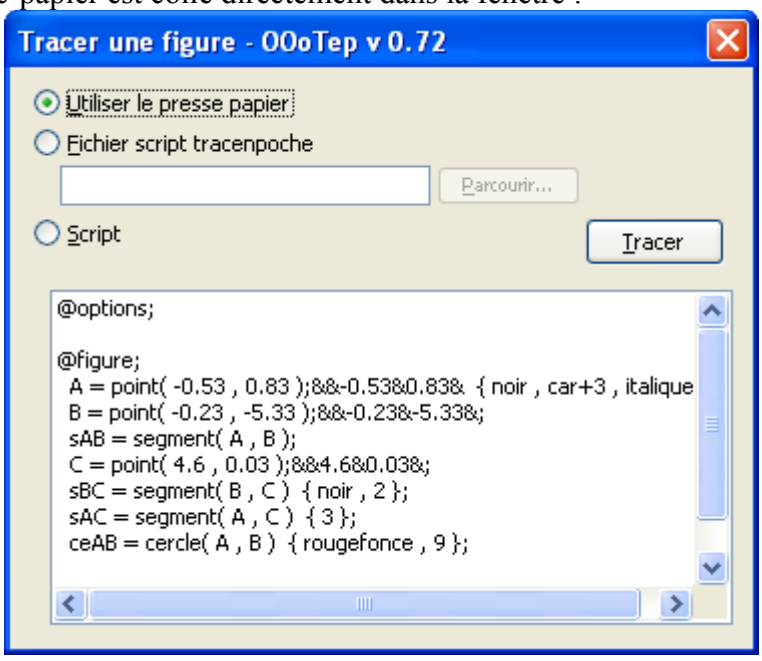
Une boîte de dialogue apparaît :



Elle permet de récupérer un script OOoTeP sauvegardé dans un fichier texte ou de coller le contenu du presse-papier.

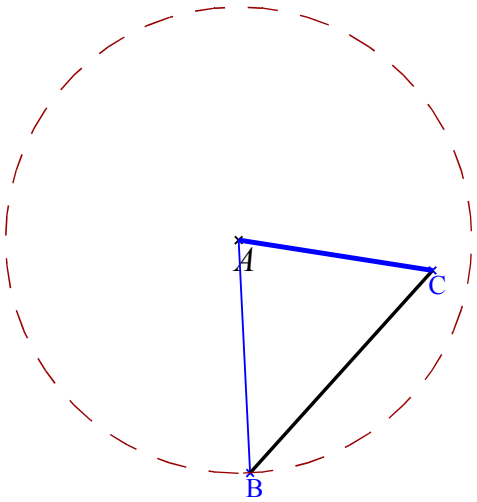
Cliquer sur le bouton radio **Utiliser le presse-papier** si vous venez de le **Copier** grâce à l'interface TracenPoche.

Le contenu du presse-papier est collé directement dans la fenêtre :



Cliquer sur le bouton **Tracer**

On obtient la figure suivante :



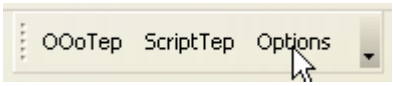
b) Modification d'une figure créée avec OOoTeP

Pour modifier une figure créée avec OOoTeP, on dispose de 2 possibilités :

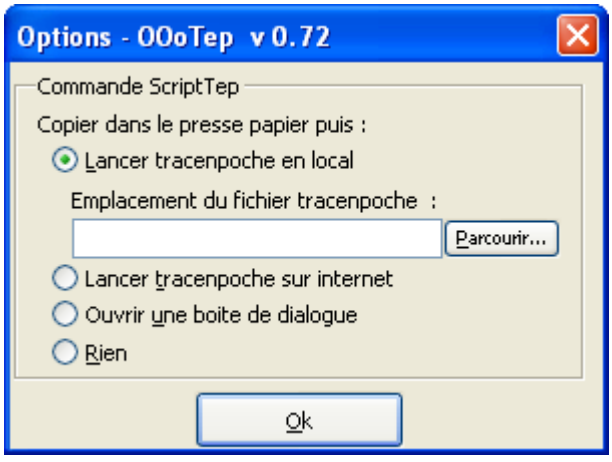
- Dégrouper la figure et la modifier à la main.
- Générer à partir de cette figure le script TeP correspondant afin de retravailler la figure dans TracenPoche lui-même !

La 1ère méthode est propre à OOo.
Détaillons plutôt la 2ème méthode.

Cliquer sur le bouton **Options** de la barre d'outils OOoTeP :



Ce bouton permet de définir ce qui se passe quand on clique sur le bouton ScriptTep alors qu'une figure OOoTeP est sélectionnée.



Il y a trois possibilités :

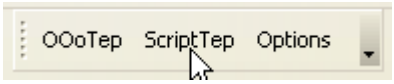
1. "Lancer TracenPoche en local" pour lancer la version de TracenPoche installée sur votre disque dur. Il faut pour cela d'indiquer le chemin de l'application juste en dessous.

2. "Lancer TracenPoche sur internet" pour lancer TracenPoche en ligne.

3. "Ouvrir une boîte de dialogue" pour générer le script TeP sous forme de texte.

Si c'est l'une des 2 premières options qui est cochée :

- **Sélectionner la figure** OOoTeP en cliquant dessus.
- Cliquer sur **ScriptTeP** dans la barre d'outils OOoTeP :



Le script TeP de la figure est copiée dans le presse-papiers, et TracenPoche se lance.

- **Coller** le script dans la `zone script` de l'interface.
- Faites les modifications sur la figure.
  Pour revenir à OOo, il faut refaire l'image par OOoTeP :
- **Générer** le script OOoTeP.
- Utiliser le bouton OOoTeP de la barre d'outils.

5. Annexes

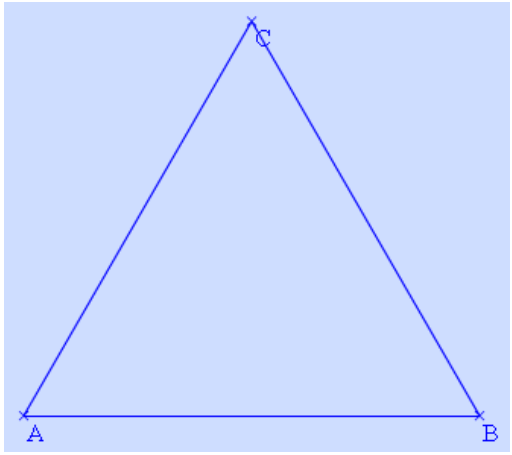
5.1 Exemples de scripts

Voici une sélection de différents scripts, du plus simple au plus compliqué - mais pas tant que ça - qui va permettre de découvrir certaines des possibilités de TracenPoche.

Ces scripts doivent servir à se familiariser avec TracenPoche, en essayant de les reproduire à l'identique. Leur but est également de donner l'inspiration pour réaliser ses propres scripts.

5.1.1 Pour commencer doucement ...

a) Exemple 1 : construction d'un triangle équilatéral.



Le point de départ de la figure est le même que sur un support papier. On trace un segment [AB], puis 2 cercles : celui de centre A passant par B et celui de centre B passant par A, ici caché pou le résultat final au moyen de l'option {i}.

```
A = point( -5 , -3 );
B = point( 5 , -3 );
sAB = segment( A , B );
ceAB = cercle( A , B )  { i };
ceBA = cercle( B , A )  { i };
```

On définit alors l'intersection de ces 2 cercles : les points C et C2. Dans ce cas, seul l'un des points est utile. Le 2ème point d'intersection et les 2 cercles sont masqués grâce à l'option `{i}` qui les rend invisibles.

```
C2 = intersection( ceBA , ceAB , 1 )  { i };
C = intersection( ceBA , ceAB , 2 );
```

Remarque : on aurait pu supprimer la ligne définissant C2 car ce point n'est plus utilisé par la suite.

La construction du triangle est terminée en traçant les 2 derniers côtés.

```
sAC = segment( A , C );
```

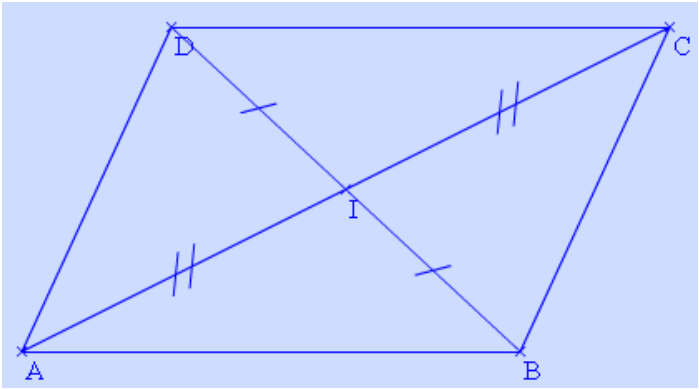
```
  sCB = segment( C , B );
```

Voici le script complet de la figure :

```
@options;

@figure;
  A = point( -5 , -3 );
  B = point( 5 , -3 );
  sAB = segment( A , B );
  ceAB = cercle( A , B )  { i };
  ceBA = cercle( B , A )  { i };
  C2 = intersection( ceBA , ceAB , 1 )  { i };
  C = intersection( ceBA , ceAB , 2 );
  sAC = segment( A , C );
  sCB = segment( C , B );
```

b) Exemple 2 : construction d'un parallélogramme

Ce parallélogramme est construit à l'aide de 3 sommets et des diagonales.



Le point de départ de la figure est le triangle ABC.

```
  A = point( -6 , -3 );
  B = point( 4 , -3 );
  sAB = segment( A , B );
  C = point( 7 , 3.5 );
  sBC = segment( B , C );
  sAC = segment( A , C );
```

Le point I est défini comme étant le milieu du segment [AC]. Le codage du milieu se fait en modifiant manuellement le script par l'ajout de l'option `{//}` au point I, ou en utilisant la palette et en cliquant sur le point I (et non sur le segment [AC]).

```
  I = milieu( sAC )  { // };
```

Le point D est construit comme étant le symétrique du point B par rapport à I.

```
  D = symetrique( B , I );
  sDC = segment( D , C );
  sAD = segment( A , D )
```

Le point I étant de fait le milieu du segment [BD] et non défini comme tel, il faut par conséquent tracer les segments [BI] et [ID] afin de les coder de manière identique.

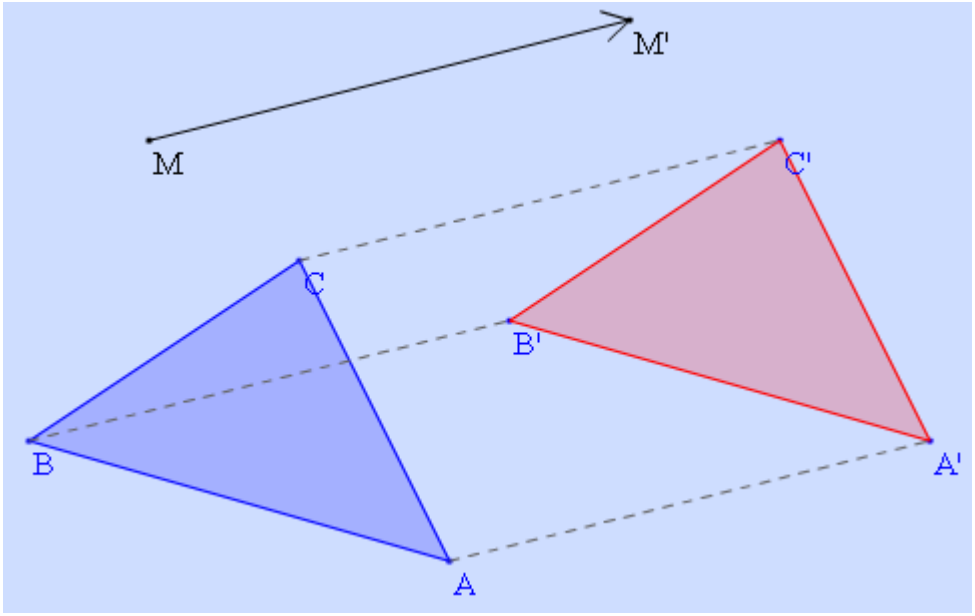
```
  sBI = segment( B , I )  { / };
  sID = segment( I , D )  { / };
```

Voici le script complet de la figure :

```
@options;

@figure;
  A = point( -6 , -3 );
  B = point( 4 , -3 );
  sAB = segment( A , B );
  C = point( 7 , 3.5 );
  sBC = segment( B , C );
  sAC = segment( A , C );
  I = milieu( sAC )  { // };
  D = symetrique( B , I );
  sDC = segment( D , C );
  sAD = segment( A , D );
  sBI = segment( B , I )  { / };
  sID = segment( I , D )  { / };
```

c) *Exemple 3 : image d'un triangle par une translation.*

Ce script a pour but de créer une image mentale de la translation en observant son effet sur un triangle.



On commence par construire les points A, B, C ainsi que le polygone ABC qui est colorié en bleu (c'est la couleur par défaut,donc plein20 suffit). Le remplissage du polygone se fait par modification manuelle du script.

```
  A = point( -1 , -5 )  { rond1 };
  B = point( -8 , -3 )  { rond1 };
  C = point( -3.5 , 0 )  { rond1 };
  polyABC = polygone( A , B , C  )  { plein20 };
```

On construit ensuite les points M et M' puis le vecteur d'origine M et d'extrémité M'.
Les points M et M' définissant le vecteur de translation sont libres, ce qui permet d'observer le comportement de la figure quand on les déplace.

```
M = point( -6 , 2 )  { noir , rond1 };
M' = point( 2 , 4 )  { noir , rond1 };
vMM' = vecteur( M , M' )  { noir };
```

On définit la translation grâce au bouton "déclarer une nouvelle transformation"  .

```
t_vMM' = translation( vMM' )  { noir };
```

Le bouton  permet de construire les points A', B' et C' images respectives des points A, B et C par la translation définie précédemment.

```
A' = image( t_vMM' , A )  { rond1 };
B' = image( t_vMM' , B )  { rond1 };
C' = image( t_vMM' , C )  { rond1 };
```

On termine en traçant le triangle A'B'C' puis en reliant en pointillés chacun des points à son image.

```
polyA'B'C' = polygone( A' , B' , C' )  { rouge , plein20 };
sCC' = segment( C , C' )  { grisfonce , 7 };
sAA' = segment( A , A' )  { grisfonce , 7 };
sBB' = segment( B , B' )  { grisfonce , 7 };
```

Voici le script complet de la figure :

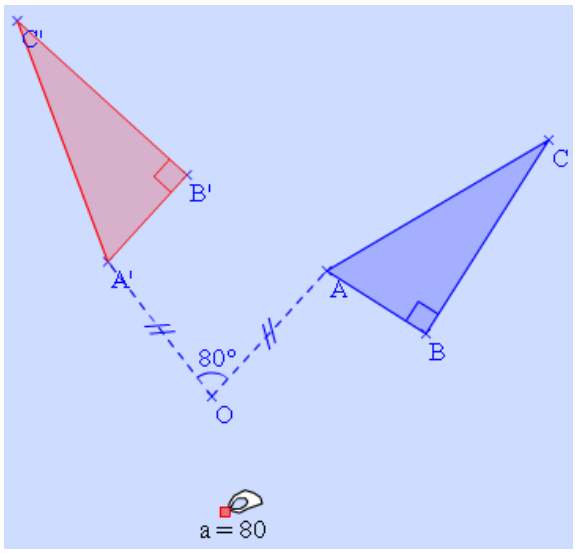
```
@options;

@figure;
  A = point( -1 , -5 )  { rond1 };
  B = point( -8 , -3 )  { rond1 };
  C = point( -3.5 , 0 )  { rond1 };
  polyABC = polygone( A , B , C )  { plein20 };
  M = point( -6 , 2 )  { noir , rond1 };
  M' = point( 2 , 4 )  { noir , rond1 };
  vMM' = vecteur( M , M' )  { noir };
  t_vMM' = translation( vMM' )  { noir };
  A' = image( t_vMM' , A )  { rond1 };
  B' = image( t_vMM' , B )  { rond1 };
  C' = image( t_vMM' , C )  { rond1 };
  polyA'B'C' = polygone( A' , B' , C' )  { rouge , plein20 };
  sCC' = segment( C , C' )  { grisfonce , 7 };
  sAA' = segment( A , A' )  { grisfonce , 7 };
  sBB' = segment( B , B' )  { grisfonce , 7 };
```

5.1.2 Pour aller un peu plus loin ...

a) *Exemple 4 : image d'un triangle par une rotation "variable".*

Ce script utilise l'objet entier. Il permet d'observer l'effet d'une rotation sur un triangle rectangle.

L'entier **a** défini dans ce script est utilisé comme angle d'une rotation. On modifie sa valeur en cliquant sur le point rouge situé au-dessus de lui et en faisant glisser la souris horizontalement tout en maintenant le bouton de la souris enfoncé.



On commence par créer le point O, puis l'entier grâce au bouton 

.

```
O = point( 1.1 , -2.93 );
a = entier( 80 , 0 , 180 , 5 )  { noir , (1,-5) };
```

On définit la rotation de centre O et d'angle a (remplacer 1 par a dans la fenêtre Angle) grâce au bouton "déclarer une nouvelle transformation" 

.

```
r_angleO = rotation( O , a )  { noir };
```

Le triangle rectangle est créé en plaçant un point C sur la perpendiculaire au segment [AB] passant par B. L'angle droit en C est codé comme tel grâce au bouton "marquage d'un angle" qui reconnaît la mesure de 90°.

```
A = point( 3.53 , -0.27 );
B = point( 5.63 , -1.6 );
sAB = segment( A , B );
perpBsAB = perpendiculaire( B , sAB )  { i };
C = pointsur( perpBsAB , 4.86 );
polyABC = polygone( A , B , C )  { plein20 };
angleABC = angle( A , B , C );
```

Le bouton  permet de construire les points A', B' et C' images respectives des points A, B et C par la rotation définie précédemment.

```
A' = image( r_angleO , A );
B' = image( r_angleO , B );
C' = image( r_angleO , C );
polyA'B'C' = polygone( A' , B' , C' )  { rouge , plein20 };
```

```
angleA'B'C' = angle( A' , B' , C' )  { rouge };
```

On termine la figure en traçant les segments [OA] et [OA'], et en marquant l'angle de sommet O.

```
sOA' = segment( O , A' )  { 7 , // };
sOA = segment( O , A )  { 7 , // };
angleA'OA = angle( A' , O , A );
```

La mesure de l'angle de la rotation est affichée grâce au bouton "mesurer l'angle défini par 3 points"  .

```
var mes_O =angle(A'OA);
t_O = texte( O , "$mes_O$°")  { dec2 };
```

Ce bouton crée 2 objets dans le script :
- une variable contenant la mesure de l'angle,
- un objet texte permettant l'affichage de la variable au sommet de l'angle.

I faut noter que, par défaut, c'est la mesure de l'angle orienté qui est définie et par conséquent affichée.
Il faut donc modifier la variable directement dans le script pour afficher la mesure de l'angle saillant non orienté : on remplace `var mes_O =A'OA` par `var mes_O =angle(A'OA);`.

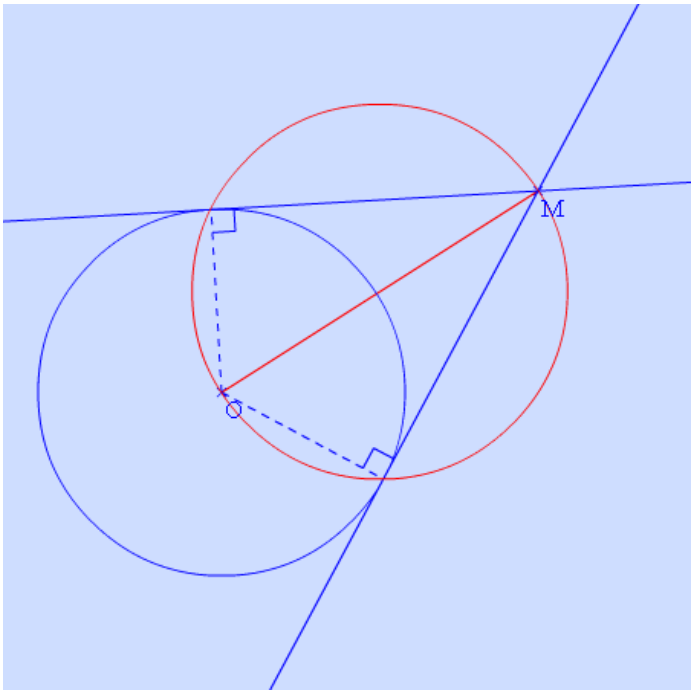
Voici le script complet de la figure :

```
@options;

@figure;
  O = point( 1.1 , -2.93 );
  a = entier( 80 , 0 , 180 , 5 )  { noir , (1,-5) };
  r_angleO = rotation( O , a )  { noir };
  A = point( 3.53 , -0.27 );
  B = point( 5.63 , -1.6 );
  sAB = segment( A , B );
  perpBsAB = perpendiculaire( B , sAB )  { i };
  C = pointsur( perpBsAB , 4.86 );
  polyABC = polygone( A , B , C  )  { plein20 };
  angleABC = angle( A , B , C );
  A' = image( r_angleO , A );
  B' = image( r_angleO , B );
  C' = image( r_angleO , C );
  polyA'B'C' = polygone( A' , B' , C'  )  { rouge , plein20 };
  angleA'B'C' = angle( A' , B' , C' )  { rouge };
  sOA' = segment( O , A' )  { 7 , // };
  sOA = segment( O , A )  { 7 , // };
  angleA'OA = angle( A' , O , A );
  var mes_O =angle(A'OA);
  t_O = texte( O , "$mes_O$°")  { dec2 };
```

b) Exemple 5 : tangentes à un cercle passant par un point.

Étant donné un cercle et un point extérieur à celui-ci, construire les tangentes au cercle passant par ce point. Voici un énoncé classique, n'est-ce pas ?

Le script suivant illustre ce problème en faisant la preuve qu'une solution existe, mais il permet également par l'utilisation d'une commande de changement état bloc de masquer la méthode de résolution.



On commence par construire le cercle de centre O et de rayon 4 (ou une autre valeur si on veut) puis un point M extérieur au cercle.

```
O = point( -3 , 0 );
cerayO4 = cerclerayon( O , 4 );
M = point( 4 , 3.5 );
```

Pour construire les 2 tangentes, la méthode utilisée est celle décrite par Euclide : les points de contact A et B des tangentes issues de M sont les points d'intersection du cercle et du cercle de diamètre [MO].

```
cediaOM = cercledia( O , M )  { rouge , i };
sOM = segment( O , M )  { rouge , i };
A = intersection( cediaOM , cerayO4 , 1 )  { i };
B = intersection( cediaOM , cerayO4 , 2 )  { i };
dMB = droite( M , B )  { sansnom };
dMA = droite( M , A )  { sansnom };
```

Noter que le cercle de diamètre [OM] et le segment [OM] sont invisibles, afin de montrer l'existence des droites tout en masquant la solution au problème de construction.
Une option de changement état bloc a donc été définie dans la section `@options;` afin de les rendre visibles en temps voulu.

```
chgt_etat_bloc("a",{v},cediaOM,sOM);
```

Pour identifier du premier coup d'oeil les tangentes, on trace les rayons et on marque les angles droits.

```
sOB = segment( O , B )  { 7 };
sOA = segment( O , A )  { 7 };
angleOBM = angle( O , B , M );
angleOAM = angle( O , A , M );
```

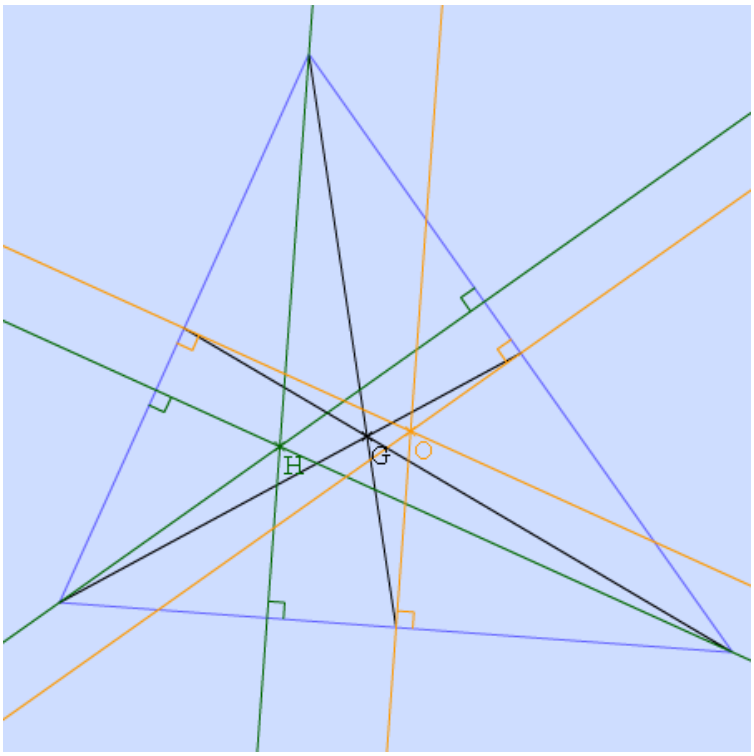
Voici le script complet de la figure :

```
@options;
```

```
  chgt_etat_bloc("a",{v},cediaOM,sOM);

@figure;
  O = point( -3 , 0 );
  cerayO4 = cerclerayon( O , 4 );
  M = point( 4 , 3.5 );
  cediaOM = cercledia( O , M )  { rouge , i };
  sOM = segment( O , M )  { rouge , i };
  A = intersection( cediaOM , cerayO4 , 1 )  { i };
  B = intersection( cediaOM , cerayO4 , 2 )  { i };
  dMB = droite( M , B )  { sansnom };
  dMA = droite( M , A )  { sansnom };
  sOB = segment( O , B )  { 7 };
  sOA = segment( O , A )  { 7 };
  angleOBM = angle( O , B , M );
  angleOAM = angle( O , A , M );
```

c) Exemple 6 : la droite d'Euler.

Le script suivant illustre l'alignement de l'orthocentre, du centre de gravité et du centre du cercle circonscrit d'un triangle, autrement dit : la droite d'Euler.



Les différentes commandes de changement état bloc permettent de ne voir que :
- soit les hauteurs,
- soit les médianes,
- soit les médiatrices,

alors que par défaut, elles sont toutes apparentes.

Une dernière commande permet de masquer ces 9 droites pour ne faire apparaître que leurs points de concours et la droite d'Euler.

On commence par construire le triangle ABC.

```
A = point( -1.5 , 7 )  { i };
B = point( -6.5 , -4 )  { i };
sAB = segment( A , B )  { bleuocean };
C = point( 7 , -5 )  { i };
sBC = segment( B , C )  { bleuocean };
sCA = segment( C , A )  { bleuocean };
```

On construit ensuite les 3 hauteurs ainsi que l'orthocentre H.

```
perpAsBC = perpendiculaire( A , sBC )  { vertfonce , sansnom };
perpBsCA = perpendiculaire( B , sCA )  { vertfonce , sansnom };
perpCsAB = perpendiculaire( C , sAB )  { vertfonce , sansnom };
H = intersection( perpAsBC , perpBsCA )  { vertfonce };
```

On construit ensuite les 3 médiatrices ainsi que le point O centre du cercle circonscrit.

```
medssBC = mediatrice( sBC )  { orange , sansnom };
medssCA = mediatrice( sCA )  { orange , sansnom };
medssAB = mediatrice( sAB )  { orange , sansnom };
O = intersection( medssCA , medssBC )  { orange };
```

On construit ensuite les 3 médianes ainsi que le point G centre de gravité. Les milieux des côtés sont nécessaires pour cette construction mais masqués.

```
C' = milieu( A , B )  { i };
A' = milieu( B , C )  { i };
B' = milieu( C , A )  { i };
sCC' = segment( C , C' )  { noir };
sBB' = segment( B , B' )  { noir };
sAA' = segment( A , A' )  { noir };
G = intersection( sCC' , sAA' )  { noir };
```

Les 3 hauteurs apparaissent seules par la combinaison **clic gauche + touche H**.

```
chgt_etat_bloc("h",{vertfonce,2},perpAsBC,perpBsCA,perpCsAB,{i},medssBC,med
               ssCA,medssAB,sAA',sCC',sBB',G,O);
```

Les 3 médiatrices apparaissent seules par la combinaison **clic gauche + touche O**.

```
chgt_etat_bloc("o",{orange,2},medssBC,medssCA,medssAB,{i},perpAsBC,perpBsCA
               ,perpCsAB,sAA',sCC',sBB',H,G);
```

Les 3 médianes apparaissent seules par la combinaison **clic gauche + touche G**.

```
chgt_etat_bloc("g",{noir,2},sAA',sCC',sBB',{i},medssBC,medssCA,medssAB,perp
               AsBC,perpBsCA,perpCsAB,H,O);
```

On termine enfin par la construction de la droite d'Euler.

```
dHO = droite( H , O )  { rougefonce , sansnom , i };
```

Cette droite est définie comme étant "invisible".
La combinaison **clic gauche + touche E** permettra de la rendre visible et de masquer les autres droites tracées précédemment.

```
  chgt_etat_bloc("e",{v,2},dHO,{i},medssBC,medssCA,medssAB,perpAsBC,perpBsCA,
                 perpCsAB,sAA',sCC',sBB');
```

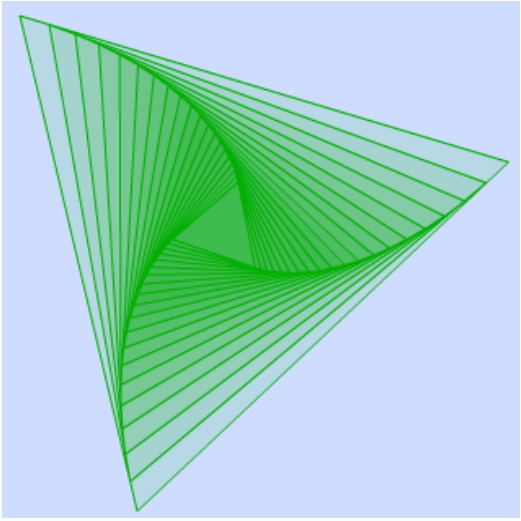
Voici le script complet de la figure :

```
@options;
  chgt_etat_bloc("h",{vertfonce,2},perpAsBC,perpBsCA,perpCsAB,{i},medssBC,med
                 ssCA,medssAB,sAA',sCC',sBB',G,O);
  chgt_etat_bloc("o",{orange,2},medssBC,medssCA,medssAB,{i},perpAsBC,perpBsCA
                 ,perpCsAB,sAA',sCC',sBB',H,G);
  chgt_etat_bloc("g",{noir,2},sAA',sCC',sBB',{i},medssBC,medssCA,medssAB,perp
                 AsBC,perpBsCA,perpCsAB,H,O);
  chgt_etat_bloc("e",{v,2},dHO,{i},medssBC,medssCA,medssAB,perpAsBC,perpBsCA,
                 perpCsAB,sAA',sCC',sBB');

@figure;
  A = point( -1.5 , 7 )  { i };
  B = point( -6.5 , -4 )  { i };
  sAB = segment( A , B )  { bleuocean };
  C = point( 7 , -5 )  { i };
  sBC = segment( B , C )  { bleuocean };
  sCA = segment( C , A )  { bleuocean };
  perpAsBC = perpendiculaire( A , sBC )  { vertfonce , sansnom };
  perpBsCA = perpendiculaire( B , sCA )  { vertfonce , sansnom };
  perpCsAB = perpendiculaire( C , sAB )  { vertfonce , sansnom };
  H = intersection( perpAsBC , perpBsCA )  { vertfonce };
  medssBC = mediatrice( sBC )  { orange , sansnom };
  medssCA = mediatrice( sCA )  { orange , sansnom };
  medssAB = mediatrice( sAB )  { orange , sansnom };
  O = intersection( medssCA , medssBC )  { orange };
  C' = milieu( A , B )  { i };
  A' = milieu( B , C )  { i };
  B' = milieu( C , A )  { i };
  sCC' = segment( C , C' )  { noir };
  sBB' = segment( B , B' )  { noir };
  sAA' = segment( A , A' )  { noir };
  G = intersection( sCC' , sAA' )  { noir };
  dHO = droite( H , O )  { rougefonce , sansnom , i };
```

5.1.3 Du travail d'expert !

a) Exemple 7 : triangles semblables et animation.

Le script suivant est très court mais son résultat est des plus jolis, surtout quand il est animé.
En voici d'ailleurs la preuve par l'image (... fixe malheureusement) :



Au premier coup d'oeil, il est évident que ce script utilise la notion de boucles, et c'est le cas.
Bien évidemment on pourrait s'en passer, mais pourquoi créer 20 triangles (et 80 lignes de script) alors qu'un seul triangle (et 14 lignes au total) suffit ?

La figure initiale est un triangle équilatéral (que vous avez déjà tracé dans l'exemple 1) dont les sommets se nomment A1, B1 et C1. Tous les points sont invisibles au final, mais pour voir/comprendre il vaut mieux les laisser visibles (en supprimant les {i}).

```
A1 = point( -4 , 4 )  { i };
B1 = point( 6 , 1 )  { i };
ceA1B1 = cercle( A1 , B1 )  { i };
ceB1A1 = cercle( B1 , A1 )  { i };
C = intersection( ceA1B1 , ceB1A1 , 1 )  { i };
C1 = intersection( ceA1B1 , ceB1A1 , 2 )  { i };
p1 = polygone( A1 , B1 , C1  )  { vert , plein10 };
```

On définit une variable réelle x qui varie entre 0 et 1 (le pas est de 0,02 mais on peut le modifier pour rendre l'animation finale plus ou moins rapide). Cette variable sera l'abscisse des points que l'on placera sur les côtés d'un triangle pour définir les sommets du triangle suivant.

```
x = reel( 0.06 , 0 , 1 , 0.02 )  { noir , oscille , (6.5,-7) , i };
```

Par l'utilisation d'une boucle on va automatiser la construction des autres triangles.

```
for i = 1 to 20 do;
A[i+1]=pointsur(A[i],B[i],x){i};
B[i+1]=pointsur(B[i],C[i],x){i};
C[i+1]=pointsur(C[i],A[i],x){i};
p[i+1]=polygone(A[i+1],B[i+1],C[i+1]){vert,plein10};
end;
```

- quand i = 1 : on place les points A2, B2 et C2 d'abscisses x sur les segments [A1B1], [B1C1] et [C1A1], puis on trace le triangle A2B2C2 nommé p2,
- quand i = 2 : on place les points A3, B3 et C3 d'abscisses x sur les segments [A2B2], [B2C2] et [C2A2], puis on trace le triangle A3B3C3 nommé p3,
- ...

jusqu'au triangle p21.

La figure peut être animée par l'appui sur le bouton "animation"  .

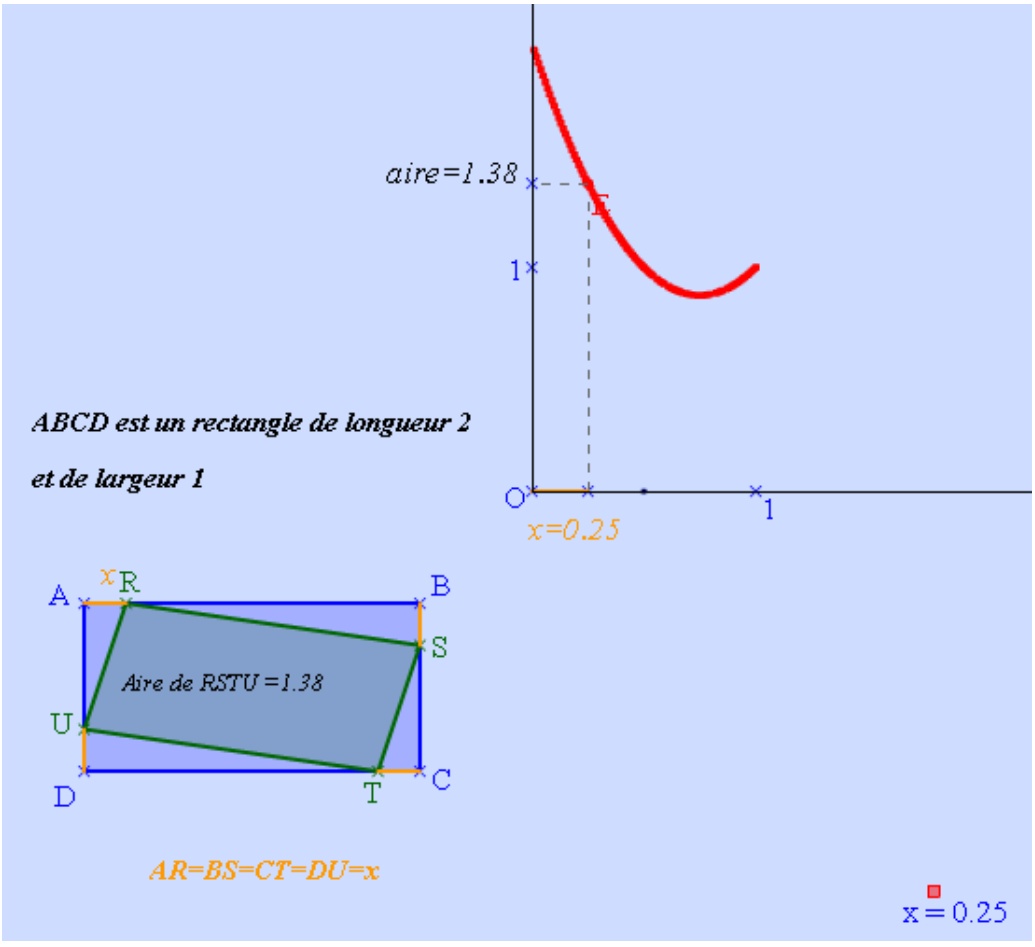
Voici le script complet de la figure :

```
@options;

@figure;
  A1 = point( -4 , 4 )  { i };
  B1 = point( 6 , 1 )  { i };
  ceA1B1 = cercle( A1 , B1 )  { i };
  ceB1A1 = cercle( B1 , A1 )  { i };
  C = intersection( ceA1B1 , ceB1A1 , 1 )  { i };
  C1 = intersection( ceA1B1 , ceB1A1 , 2 )  { i };
  p1 = polygone( A1 , B1 , C1  )  { vert , plein10 };
  x = reel( 0.06 , 0 , 1 , 0.02 )  { noir , oscille , (6.5,-6.96) , i };
  for i = 1 to 20 do;
  A[i+1]=pointsur(A[i],B[i],x){i};
  B[i+1]=pointsur(B[i],C[i],x){i};
  C[i+1]=pointsur(C[i],A[i],x){i};
  p[i+1]=polygone(A[i+1],B[i+1],C[i+1]){vert,plein10};
  end;
```

b) Exemple 8 : un problème d'optimisation.

Énoncé du problème :

ABCD est un rectangle de dimensions 1 et 2.
R, S ,T et U appartiennent aux segments [AB], [BC], [CD] et [DA] tel que AR=BS=CT=DU= *x*.
Pour quelle valeur de *x*, l'aire de RSTU est elle minimale ?



Le script réalisé va permettre, en cliquant sur le bouton "animation"  , de tracer la courbe représentant l'aire de RSTU en fonction de *x*.

1ère étape :

On construit le rectangle ABCD. Pour ne pas passer trop de temps à le construire, on "triche" en utilisant les coordonnées des points (on peut éventuellement ajouter de manière temporaire les commandes **`trame();`** et **`aimante();`** dans la section **`@options;`**). On ajoute l'attribut **`fixe`** à chacun des points afin que ce rectangle ne puisse pas être "déformé".

```
A = point( -8 , -2 )  { fixe , (-0.69,-0.54) };
B = point( -2 , -2 )  { fixe , (0.12,-0.69) };
C = point( -2 , -5 )  { fixe , (0.12,-0.24) };
D = point( -8 , -5 )  { fixe , (-0.6,0.08) };
polyABCD = polygone( A , B , C , D  )  { 2 , plein20 };
```

On définit le réel *x* qui varie entre 0 et 1 afin de placer les points sur les côtés du rectangle. L'option oscille est ajoutée afin d'animer la figure par la suite.

```
x = reel( 0.75 , 0 , 1 , 0.01 )  { oscille , (7.27,-7.47) };
```

Dans la syntaxe **`R = pointsur( A , B , x );`** *x* est l'abscisse du point R sur l'axe gradué d'origine A, d'unité AB et d'orientation de A vers B . Par conséquent : AR = *x*×AB

Les côtés [BC] et [AD] mesurant 1cm, en définissant les points S et U avec comme abscisse *x* sur chacun de ces segments, on aura bien : BS = DU = *x*.

Le problème se pose pour les points R et T car les segments sur lesquels ils sont placés mesurent 2 cm.
En effet si on définit le point R comme ceci : **`R = pointsur( A , B , x );`**, alors on AR = 2*x*.

L'astuce est donc de créer une 2ème variable *a* égale à *x*/2.

```
var a =x/2 ;
```

Il ne reste plus qu'à placer les points sur chacun des côtés, en modifiant dans le script l'abscisse de chacun : on la remplace par *x* ou *a* selon les côtés sur lesquels ils se trouvent.

```
R = pointsur( A , B , a )  { vertfonce , (-0.2,-0.78) };
S = pointsur( B , C , x )  { vertfonce , (0.12,-0.34) };
T = pointsur( C , D , a )  { vertfonce , (-0.31,0) };
U = pointsur( D , A , x )  { vertfonce , (-0.66,-0.5) };
po = polygone( R , S , T , U  )  { vertfonce , 2 , plein20 };
```

On stocke dans une variable l'aire de RSTU que l'on affiche à l'aide d'un objet texte.

```
var aire =2x*x-3x+2 { 0.875 };
texte1 = texte( -7.4 , -3.1 , "Aire de RSTU =$aire$")  { noir ,
                                                dec2 , car-2 , italique };
```

2ème étape :

On construit le repère (O,I,J) et les demi-droites [OI) et [OJ).

```
O = point( 0 , 0 )  { (-0.58,-0.26) };
I = point( 4 , 0 )  { sansnom };
J = point( 0 , 4 )  { sansnom };
demiOI = demidroite( O , I )  { noir , sansnom };
demiOJ = demidroite( O , J )  { noir , sansnom };
```

Sur la demi-droite [OI) on place le point M d'abscisse *x*, et sur la demi-droite [OJ) on place le point N d'abscisse l'aire de RSTU.

```
M = pointsur( O , I , x )  { sansnom };
N = pointsur( O , J , aire )  { sansnom };
```

Par chacun de ses points on trace les perpendiculaires aux axes (elles sont masquées ensuite), à l'intersection se trouve le point E, qui dans le repère (O,I,J) a donc pour coordonnées : (*x* ; aire de RSTU).

```
perpNdemiOJ = perpendiculaire( N , demiOJ )  { i };
perpMdemiOI = perpendiculaire( M , demiOI )  { i };
E = intersection( perpNdemiOJ , perpMdemiOI )  { rouge , trace ,
                                                          rond2 };
sME = segment( M , E )  { grisfonce , 7 };
sEN = segment( E , N )  { grisfonce , 7 };
```

3ème étape :

On termine en soignant la mise en forme de la figure.

Une astuce à utiliser régulièrement avec TracenPoche, c'est d'attacher un texte à un point dont on a masqué le nom.
On affiche donc la valeur de *x* à la place de M, l'aire de RSTU à la place de N et 1 à la place des points I et J.

```
t2 = texte( M , "x=$x$")  { orange , (-1.13,0.3) , dec2 , italique };
t3 = texte( N , "aire=$aire$")  { noir , (-2.66,-0.58) , dec2 ,
italique };
t4 = texte( I , "1")  { (0.04,-0.06) , dec2 };
t5 = texte( J , "1")  { (-0.5,-0.32) , dec2 };
```

La fin du script permet de résumer la situation géométrique dans laquelle on se trouve, ainsi que de mettre en forme RSTU.

```
texte2 = texte( -9 , 1.6 , "ABCD est un rectangle de longueur 2")
                              { noir , dec2 , car-1 , gras,italique };
texte3 = texte( -9 , 0.6 , "et de largeur 1")  { noir , dec2 ,
                                        car-1 , gras , italique };
texte4 = texte( -6.9 , -6.4 , "AR=BS=CT=DU=x")  { orange , dec2 ,
                                        car-1 , gras , italique };
sAR = segment( A , R )  { orange , 2 };
sBS = segment( B , S )  { orange , 2 };
sCT = segment( C , T )  { orange , 2 };
sDU = segment( D , U )  { orange , 2 };
L = milieu( A , R )  { croix0 , sansnom };
t6 = texte( L , "x")  { orange , (-0.14,-0.89) , dec2 , italique };
sOM = segment( O , M )  { orange , 2 };
```

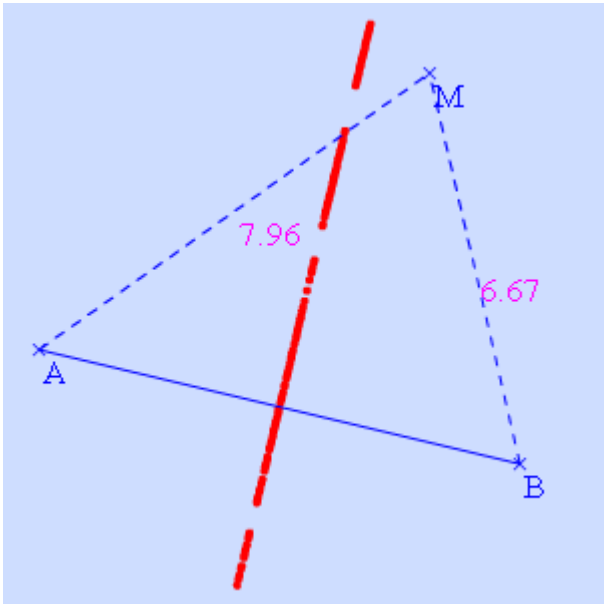
Pour répondre à la question posée, il suffit d'animer la figure et d'observer la trace laissée par le point E.

Voici le script complet de la figure :

```
@options;
  repereortho(310,270,30,1,1){ 0 , moyen , noir , num1 ,i};

@figure;
  A = point( -8 , -2 )  { fixe , (-0.69,-0.54) };
  B = point( -2 , -2 )  { fixe , (0.12,-0.69) };
  C = point( -2 , -5 )  { fixe , (0.12,-0.24) };
  D = point( -8 , -5 )  { fixe , (-0.6,0.08) };
  polyABCD = polygone( A , B , C , D  )  { 2 , plein20 };
  x = reel( 0.75 , 0 , 1 , 0.01 )  { oscille , (7.27,-7.47) };
  var a =x/2 { 0.375 };
  R = pointsur( A , B , a )  { vertfonce , (-0.2,-0.78) };
  S = pointsur( B , C , x )  { vertfonce , (0.12,-0.34) };
  T = pointsur( C , D , a )  { vertfonce , (-0.31,0) };
  U = pointsur( D , A , x )  { vertfonce , (-0.66,-0.5) };
  po = polygone( R , S , T , U  )  { vertfonce , 2 , plein20 };
  var aire =2x*x-2x+2 { 1.625 };
  texte1 = texte( -7.4 , -3.1 , "Aire de RSTU =$aire$")  { noir ,
                                                dec2 , car-2 , italique };
  O = point( 0 , 0 )  { (-0.58,-0.26) };
  I = point( 4 , 0 )  { sansnom };
  J = point( 0 , 4 )  { sansnom };
  demiOI = demidroite( O , I )  { noir , sansnom };
  demiOJ = demidroite( O , J )  { noir , sansnom };
  K = point( 2 , 0 )  { bleufonce , rond1 , sansnom };
  M = pointsur( O , I , x )  { sansnom };
  N = pointsur( O , J , aire )  { sansnom };
  perpNdemiOJ = perpendiculaire( N , demiOJ )  { i };
  perpMdemiOI = perpendiculaire( M , demiOI )  { i };
  E = intersection( perpNdemiOJ , perpMdemiOI )  { rouge , trace ,
                                                          rond2 };
  sME = segment( M , E )  { grisfonce , 7 };
  sEN = segment( E , N )  { grisfonce , 7 };
  t2 = texte( M , "x=$x$")  { orange , (-1.13,0.3) , dec2 , italique };
  t3 = texte( N , "aire=$aire$")  { noir , (-2.66,-0.58) , dec2 ,
                                                    italique };
  t4 = texte( I , "1")  { (0.04,-0.06) , dec2 };
  t5 = texte( J , "1")  { (-0.5,-0.32) , dec2 };
  texte2 = texte( -9 , 1.6 , "ABCD est un rectangle de longueur 2")
                                { noir , dec2 , car-1 , gras,italique };
  texte3 = texte( -9 , 0.6 , "et de largeur 1")  { noir , dec2 ,
                                                car-1 , gras,italique };
  texte4 = texte( -6.9 , -6.4 , "AR=BS=CT=DU=x")  { orange , dec2 ,
                                                car-1 , gras,italique };
  sAR = segment( A , R )  { orange , 2 };
  sBS = segment( B , S )  { orange , 2 };
  sCT = segment( C , T )  { orange , 2 };
  sDU = segment( D , U )  { orange , 2 };
  L = milieu( A , R )  { croix0 , sansnom };
  t6 = texte( L , "x")  { orange , (-0.14,-0.89) , dec2 , italique };
  sOM = segment( O , M )  { orange , 2 };
```

c) *Exemple 9 : découverte de la médiatrice.*

Ce script, qui permet de découvrir la médiatrice comme lieu de point, va servir ici d'exemple d'utilisation des points aimantés et de la fonction µ pour une mise en forme conditionnelle.



Pour commencer, on construit le segment [AB] et sa médiatrice que l'on rend invisible.

```
A = point( -3 , 0 );
B = point( 4.5 , -1 );
sAB = segment( A , B );
medsAB = mediatrice( sAB )  { i };
```

On crée le point M qui doit être aimanté sur la médiatrice de [AB]. Ce point doit être créé directement dans le script. On choisit une précision de 10% ce qui signifie que lorsque la distance entre le point M et la médiatrice sera inférieure ou égale à 0,1, alors le point M se positionnera sur la médiatrice.

```
M = pointaimante( 3 , 3 , M appartient medsAB_10% );
```

Quand le point M est sur la médiatrice de [AB] on veut que celui-ci laisse une trace. Il est impossible de créer une mise en forme conditionnelle pour le point M car il intervient dans la condition. L'astuce est donc d'utiliser un autre point, confondu avec M et qui sera celui dont on verra la trace. Ce point est le symétrique de M par rapport à M, donc, M lui-même.
Pour la mise en forme conditionnelle on utilise un objet varsi.

```
varsi x =[MappartientmedsAB,"rouge-trace","bleu-pastrace"] ;
```

On construit M1, symétrique de M par rapport à M, on masque son nom, et on ajoute la variable x dans ses options de mise en forme.

```
M1 = symetrique( M , M )  {$x$ , sansnom };
```

Si le point M n'appartient pas à la médiatrice de [AB], x prend pour valeur "bleu-pastrace", le point M1 est alors invisible à nos yeux car confondu avec M mais sans nom. Par contre, si M appartient à la médiatrice de [AB], alors x prend pour valeur "rouge-trace", ce qui implique que le point M1 laisse une trace rouge.

On termine ensuite le script en traçant les segments [MA] et [MB], puis en affichant leurs longueurs.

```
  sAM = segment( A , M )  { 7 };
  sBM = segment( B , M )  { 7 };
  var disAM =AM { 6.6113650044618 };
  p_disAM = milieu( A , M )  { i };
  t_disAM1 = texte( p_disAM ,"$disAM$")  { magenta , dec2 };
  var disBM =BM { 4.67341565548036 };
  p_disBM = milieu( B , M )  { i };
  t_disBM1 = texte( p_disBM ,"$disBM$")  { magenta , dec2 };
```

Voici le script complet de la figure :

```
@options;

@figure;
  A = point( -3.43 , 0.77 );
  B = point( 4.57 , -1.13 );
  sAB = segment( A , B );
  medsAB = mediatrice( sAB )  { i };
  M = pointaimante( 2.73 , 3.17 , M appartient medsAB_10% );
  varsi x =[MappartientmedsAB,"rouge-trace","bleu-pastrace"];
  M1 = symetrique( M , M )  {$x$ , sansnom };
  sAM = segment( A , M )  { 7 };
  sBM = segment( B , M )  { 7 };
  var disAM =AM { 6.6113650044618 };
  p_disAM = milieu( A , M )  { i };
  t_disAM1 = texte( p_disAM ,"$disAM$")  { magenta , dec2 };
  var disBM =BM { 4.67341565548036 };
  p_disBM = milieu( B , M )  { i };
  t_disBM1 = texte( p_disBM ,"$disBM$")  { magenta , dec2 };
```

5.2 Créer une activité au format html en utilisant TepWeb

En allant sur le site de TracenPoche : www.tracenpoche.net, on peut visiter la rubrique **Activithèque**.
Et en cliquant sur les différents liens proposés, on se dit peut-être : est-ce compliqué de faire aussi bien, si ce n'est mieux ...?

Les pages qui suivent vont montrer que ceci est à la portée de tous, et qu'il n'est nul besoin de posséder tel ou tel logiciel pour créer de jolies pages. On va se contenter de l'éditeur de textes le plus simple qui existe sur tout ordinateur. Car un bloc-notes suffit amplement. Si ensuite on veut faire plus de fantaisies, on pourra toujours utiliser un logiciel d'édition de pages Web comme NVu qui est libre et gratuit à défaut de maîtriser le langage html ;-).

1ère étape : réalisation de la figure avec TeP.

La figure qui va servir de support pour la réalisation de notre page Web, est une des figures clés de la classe de 5ème.
Elle est constituée de 2 droites parallèles et d'une sécante. Une animation permet de visualiser le centre de symétrie de cette figure.

En voici le script :

```
@options;

@figure;
  A = point( -4 , -3 )  { i };
  B = point( 7 , -1.5 )  { i };
  C = point( 2 , 5 )  { i };
  dAB = droite( A , B )  { sansnom };
  paraCdAB = parallele( C , dAB )  { sansnom };
  dAC = droite( A , C )  { sansnom };
  O = milieu( C , A )  { // };
  a = reel( 0 , 0 , 180 , 1 )  { oscille , (8,-8) , i };
  r_angleO = rotation( O , a )  { noir };
  A' = image( r_angleO , A )  { i };
  B' = image( r_angleO , B )  { i };
  C' = image( r_angleO , C )  { i };
  dA'B' = droite( A' , B' )  { 7 , sansnom };
  paraC'dA'B' = parallele( C' , dA'B' )  { 7 , sansnom };
  dA'C' = droite( A' , C' )  { 7 , sansnom };
```

La figure étant chargée grâce à l'interface TepWeb dans notre future page Web, il n'est pas nécessaire que le fichier figure contienne toutes les sections. Les sections `@enonce` et `@analyse` sont ici inutiles. Par contre la section `@config` ne sera pas vide.
En effet on désire pouvoir animer la figure et donc le bouton "animation"  doit apparaître dans la fenêtre TepWeb.
Pour cela, ajouter ces lignes au script de la figure :

```
@config
  boutons=animation;
```

Le script complet de la figure à enregistrer au format txt contient donc ceci :

```
@options;

@figure;
  A = point( -4 , -3 )  { i };
  B = point( 7 , -1.5 )  { i };
  C = point( 2 , 5 )  { i };
  dAB = droite( A , B )  { sansnom };
  paraCdAB = parallele( C , dAB )  { sansnom };
  dAC = droite( A , C )  { sansnom };
  O = milieu( C , A )  { // };
  a = reel( 0 , 0 , 180 , 1 )  { oscille , (8,-8) , i };
  r_angleO = rotation( O , a )  { noir };
  A' = image( r_angleO , A )  { i };
  B' = image( r_angleO , B )  { i };
  C' = image( r_angleO , C )  { i };
  dA'B' = droite( A' , B' )  { 7 , sansnom };
  paraC'dA'B' = parallele( C' , dA'B' )  { 7 , sansnom };
  dA'C' = droite( A' , C' )  { 7 , sansnom };

@config;
  boutons=animation;
```

Nommer ce fichier **figure.txt**, et l'enregistrer dans un répertoire nommé **html**.

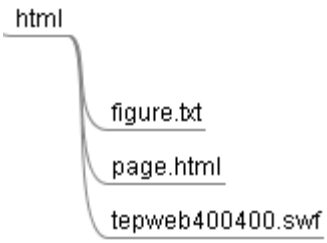
La figure est maintenant prête à être insérée dans une page Web.

2ème étape : préparation des autres fichiers.

A l'intérieur du répertoire **html** on va créer un nouveau document texte.
Enregistrer ce fichier sous le nom **page.html**. Son aspect a changé, l'icône qui le représente est maintenant celle du navigateur Web par défaut.

A l'intérieur du répertoire **html**, coller le fichier **TepWeb** qui servira à afficher la figure dans notre page web. Si on veut, par exemple, afficher la figure dans une zone de 400 pixels par 400 pixels, c'est donc du fichier **tepweb400400.swf** dont on aura besoin ici.

Le contenu du répertoire **html** devrait ressembler à ceci :



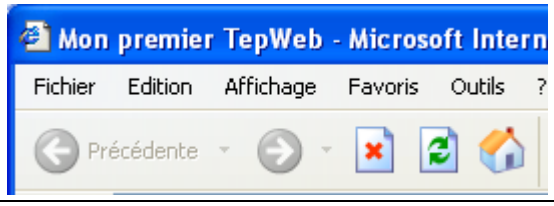
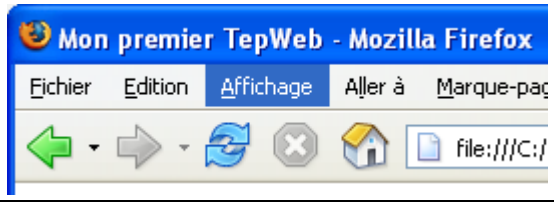
3ème étape : création de la page Web.

Ouvrir le fichier **page.html**. Il faut noter que si on l'ouvre en effectuant un double-clique, il s'ouvre par l'intermédiaire du navigateur internet. Le plus simple est donc de l'ouvrir en effectuant un clic droit, puis de sélectionner "Ouvrir avec ..." et de choisir dans la liste l'éditeur de textes.

A l'intérieur de ce document taper ceci :

```
<html>
<head>
<title>Mon premier TepWeb</title>
</head>
<body>
</body>
</html>
```

Maintenant, ouvrir le fichier **page.html** avec son navigateur préféré : la page est vide mais son titre a été modifié.

Avec MS Interner Explorer	Avec Mozilla Firefox
	

Les balises <html> et </html> indique le début et la fin d'un document écrit en langage html.

Les balises <head> et </head> indique le début et la fin de la zone d'en-tête de la page (head signifie tête en anglais). C'est une zone prologue au document proprement dit, elle permet de donner certains renseignements au navigateur.

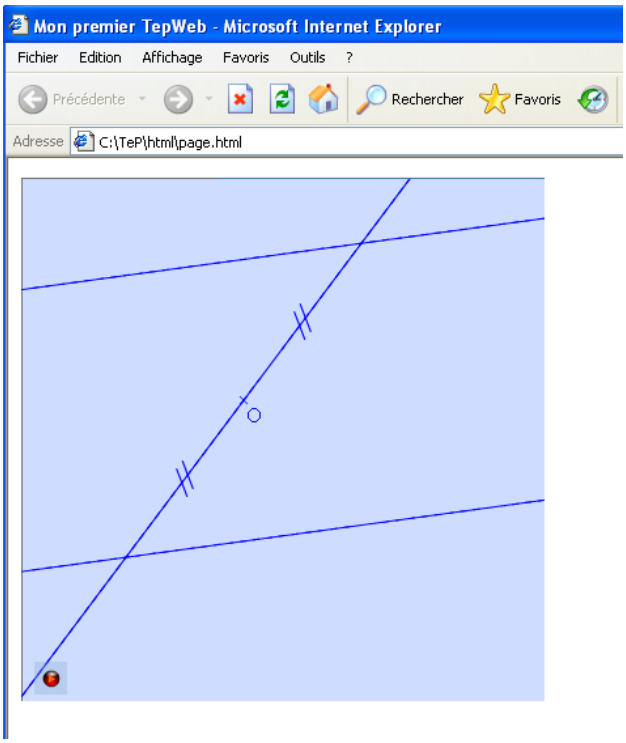
Les balises <title> et </title> indique le début et la fin du titre de la page (title signifie titre en anglais). On remarque que c'est ce que l'on a écrit entre ces 2 balises qui apparaît comme titre dans la fenêtre du navigateur.

Les balises <body> et </body> indique le début et la fin du document proprement dit (body signifie corps en anglais). Le document est vide car, pour l'instant, il n'y a rien entre ces 2 balises.

Pour insérer la figure TepWeb à l'intérieur de cette page, il suffit d'ajouter le texte suivant dans le corps du document, entre les balises <body> et </body> :

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/s
  wflash.cab#version=7,0,0,0" width="400" height="400" align="middle">
     <param name="allowScriptAccess" value="sameDomain" />
     <param name="quality" value="high" />
     <param name="bgcolor" value="#ffffff" />
     <param name="SRC" value="tepweb400400.swf?script=figure.txt">
     <embed src="tepweb400400.swf?script=figure.txt" width="400"
       height="400" align="middle" quality="high" bgcolor="#ffffff"
       swLiveConnect=true id="figure" allowScriptAccess="sameDomain"
       type="application/x-shockwave-flash"
       pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

Enregistrer le document et visualiser le résultat dans son navigateur.



On obtient le résultat attendu, la figure s'affiche dans une fenêtre de 400 pixels par 400 pixels, et s'anime quand on clique sur le bouton "animation" en bas à gauche.

On a donc fini ... ou presque !

4ème étape : une finition de "pro".

Pour "embellir" encore plus ce document, on va ajouter une question auquel l'élève répondra en complétant un champ de texte.

La question posée se présentera sous la forme d'une phrase à trou : La figure admet un ..... de symétrie.

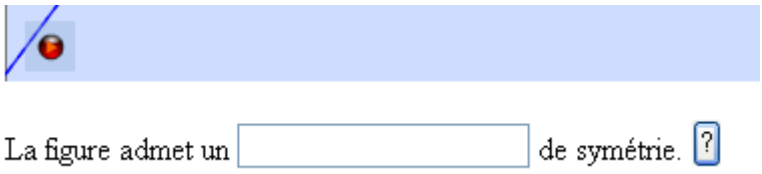
Après avoir complété la phrase, l'élève devra appuyer sur un bouton pour vérifier que sa réponse est correcte.

Pour effectuer cette "correction", on va utiliser ce que l'on appelle un script, tout comme on programme une figure dans TracenPoche, on va ici programmer un peu. Le langage utilisé pour créer ce script est le Javascript. Javascript est une extension du code html des pages Web. Les scripts, qui s'ajoutent ici aux balises html, peuvent en quelque sorte être comparés aux macros d'un tableur ou d'un traitement de texte : ici on télécommande le navigateur.

Commencer par ajouter les lignes suivantes dans le document **page.html** à la suite de la déclaration de l'objet TepWeb.

```
<br>
<form name="form1">
  La figure admet un
  <input type="text" name="reponse">
  de symétrie.
  <input name="bouton" type="button" value="?" onClick="verif()">
</form>
```

Ces lignes permettent de créer la phrase à compléter ainsi que le bouton qui lancera la vérification de la réponse.



La balise **
** permet de créer un passage à la ligne.

Les balises **<form>** et **</form>** permettent de créer un formulaire html, ce formulaire se nomme "form1" (**name="form1"**). La création du formulaire est indispensable pour pouvoir récupérer le contenu du champ de texte.

A l'intérieur de ce formulaire, on crée un champ de texte à compléter (**<input type="text" name="reponse">**).
Le champ de texte s'appelle "reponse" (sans accent !) : c'est son contenu que l'on vérifiera grâce au script.

La commande **<input name="...>** permet d'insérer un bouton dont le nom est "bouton", l'étiquette (c'est à dire ce qui s'affiche sur le bouton) est un ? et la commande **onClick="verif()"** indique qu'en cliquant sur le bouton on lance la routine du script nommé verif().

C'est donc ce script qu'on va créer. Pour cela, insérer les lignes suivantes entre les balises **<head>** et **</head>** juste après le titre de la page.

```
<script language="JavaScript">
function verif() {
      if (document.form1.reponse.value=="centre") {
            window.alert('Bonne réponse');
      } else {
            window.alert('Mauvaise réponse');
      }
}
</script>
```

La première ligne indique que le langage utilisé est du Javascript.

On déclare ensuite la création d'une nouvelle routine, une fonction ("function") nommée verif().

Celle-ci teste par un if (si en anglais) le contenu du champ de texte nommé "reponse" situé dans le formulaire "form1" :

- si ce champ contient le mot *centre* alors une fenêtre s'ouvre indiquant que la réponse est bonne,
- sinon (else), le contenu est différent alors, une fenêtre s'ouvre indiquant que la réponse est mauvaise.

Si la réponse donnée est "centre"	Si la réponse est donnée est autre que "centre"
	

Le contenu complet de la page est maintenant :

```
<html>
<head>
<title>Mon premier TepWeb</title>
<script language="JavaScript">
function verif() {
      if (document.form1.reponse.value=="centre") {
            window.alert('Bonne réponse');
      } else {
            window.alert('Mauvaise réponse');
      }
}
</script>
</head>
```

```
<body>
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/s
  wflash.cab#version=7,0,0,0" width="400" height="400" align="middle">
<param name="allowScriptAccess" value="sameDomain" />
<param name="quality" value="high" />
<param name="bgcolor" value="#ffffff" />
<param name="SRC" value="tepweb400400.swf?script=figure.txt">
<embed src="tepweb400400.swf?script=figure.txt" width="400"
  height="400" align="middle" quality="high" bgcolor="#ffffff"
  swLiveConnect=true id="figure" allowScriptAccess="sameDomain"
  type="application/x-shockwave-flash"
  pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
<br>
<form name="form1">
  La figure admet un
  <label>
  <input type="text" name="reponse">
  </label>
  de symétrie.
  <input name="bouton" type="button" value="?" onClick="verif()">
</form>
</body>
</html>
```

Et maintenant, libre à chacun de découvrir par soi même toutes les possibilités qu'offre l'utilisation combinée de TracenPoche et du Javascript.

5.3 Utilisation de TeP dans les exercices de MeP Réseau

5.3.1 Présentation

MathenPoche est un logiciel d'exercices de mathématiques à destination des élèves, en accès libre et gratuit sur internet : www.MathenPoche.net .
L'élève peut ainsi l'utiliser à domicile comme en classe, sous la direction d'un professeur.

La version réseau de MathenPoche peut être installée sur un serveur internet (voir le serveur national implanté en Seine-et-Marne dédié à MathenPoche) ou en intranet via un réseau local.

Pour utiliser cette version, l'élève doit saisir un nom d'utilisateur et un mot de passe qui lui sont personnels. Le logiciel identifie alors l'élève connecté, ce qui permet de personnaliser l'accès aux exercices, de récupérer et de traiter les résultats.

Le professeur intervient lui aussi sur un ordinateur. Une interface lui est réservée : elle permet en quelques clics de souris de créer ou modifier par avance des groupes d'élèves et des séances personnalisées, de suivre la progression des élèves en temps réel, d'analyser les résultats pour affiner son enseignement.

Une banque d'exercices est à la disposition du professeur pour la préparation de sa séance. Parmi ces exercices, certains utilisent déjà TracenPoche par l'intermédiaire de l'interface TepNoyau (voir **4.2 TepNoyau**).

Depuis le mois de septembre 2005, la possibilité est donnée aux professeurs de créer leurs propres activités TracenPoche afin de les insérer dans une séance personnalisée MathenPoche.
Le professeur récupérera le travail de l'élève sous la forme d'un script, celui de la figure finale, mais évidemment sans évaluation a priori

5.3.2 Créer et utiliser un exercice TeP dans MeP réseau.

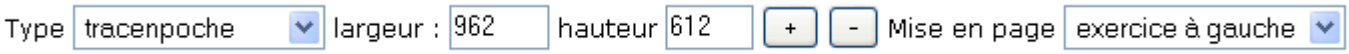
- **Créer un exercice.**

Dans l'interface formateur de MathenPoche réseau, la rubrique "Outils" permet d'accéder à l'interface de création d'exercices TeP.

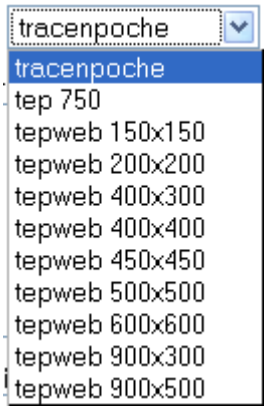
En cliquant sur "Exos TracenPoche", on accède à l'interface de création d'exercice TeP à insérer dans la séance MeP réseau.

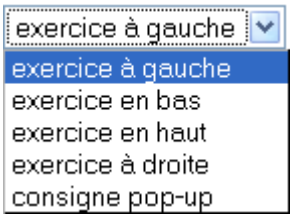
Dans la première partie de l'interface il est demandé de donner un titre à l'exercice ainsi que de donner des mots clés qui permettront par la suite de retrouver facilement l'exercice.

La deuxième partie de l'interface permet de choisir la mise en page de l'exercice.

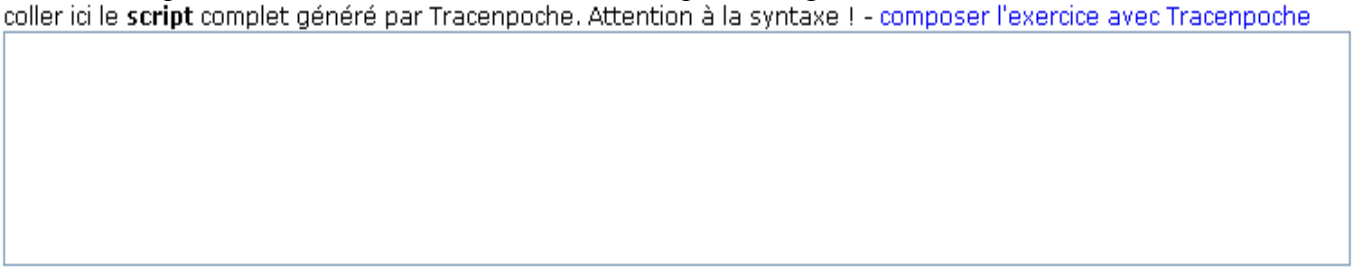


On peut en effet choisir d'intégrer dans l'exercice, soit l'interface TracenPoche, soit l'interface TepWeb. Mais on peut également choisir la taille de l'interface ainsi que sa position par rapport à l'énoncé de l'exercice.





La troisième partie de l'interface sert à recevoir le script de la figure.



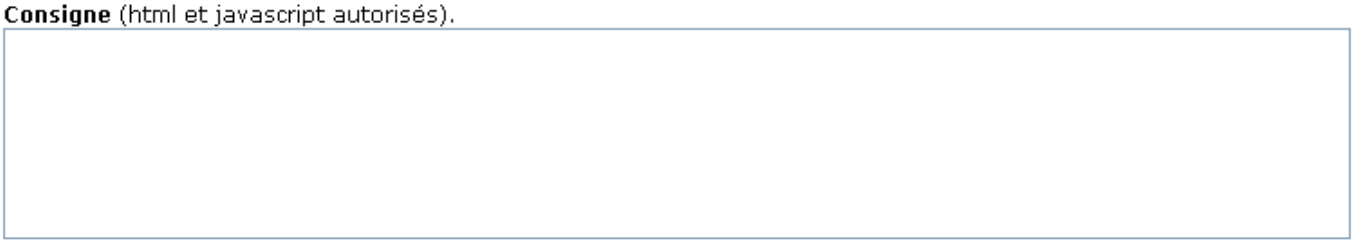
Le script est celui généré par l'interface TeP, on peut bien évidemment le modifier à sa guise selon les besoins de l'exercice :

- choisir les boutons à afficher en modifiant la section @config pour une interface TeP
- changer la couleur de fond de la zone de dessin pour TepWeb
- ...

Attention ! si on insère l'interface TracenPoche, l'élève doit pouvoir appuyer sur le bouton **tepmep** 

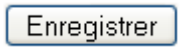
pour pouvoir par la suite accéder au script de sa figure. Donc ce bouton doit apparaître dans la liste des boutons disponibles.

La partie "**Consigne**" de l'interface sert à y mettre la consigne qui sera affichée à coté, au dessus, en dessous ou dans une fenêtre PopUp.

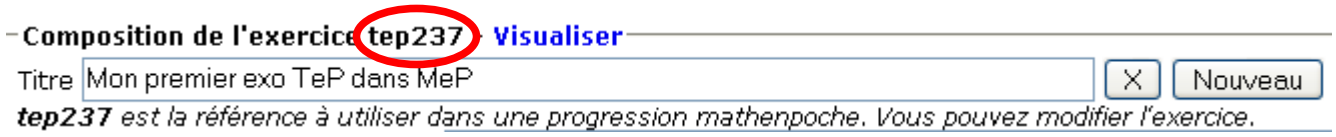


On peut utiliser du code HTML ou du Javascript afin, par exemple, de modifier la mise en forme du texte ou d'insérer une zone de texte.

Dans la partie "**Commentaire**" on peut mettre un descriptif de l'exercice qui sera affiché lors de la sélection de l'exercice au moment de la préparation de la séance MeP.

Il suffit alors d'enregistrer cet exercice : 

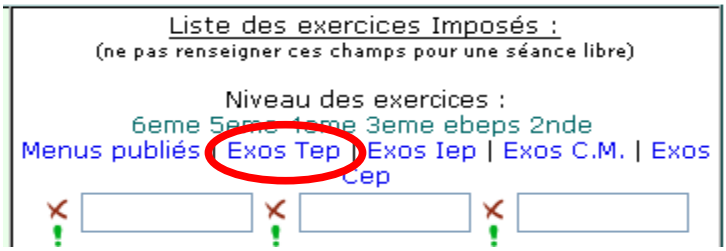
Une fois l'exercice enregistré, une référence lui est attribuée, cette référence permettra par la suite de l'insérer dans une séance MeP réseau.



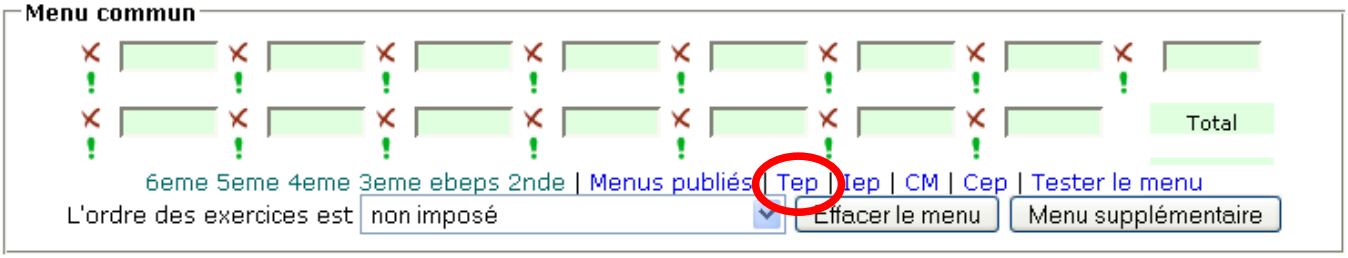
- **Insérer un exercice TeP dans une séance MeP**

Lors de la création d'une séance MeP ("séance rapide" ou "nouvelle séance"), en dehors des exercices du logiciel MeP la possibilité est donnée d'insérer des "Exos TeP".

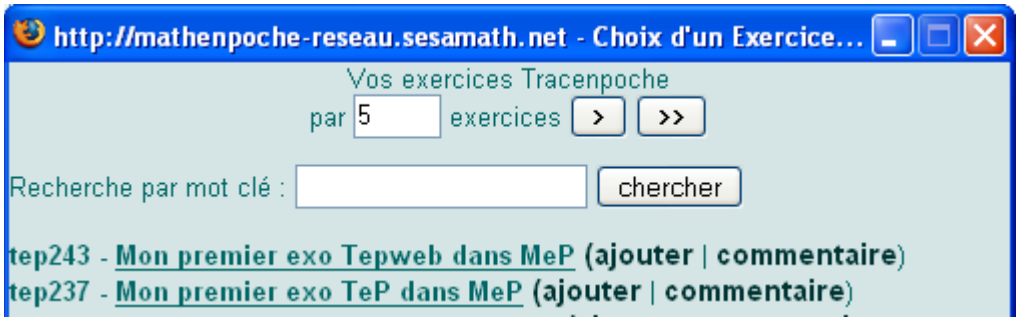
Pour une séance rapide :



Pour une nouvelle séance :

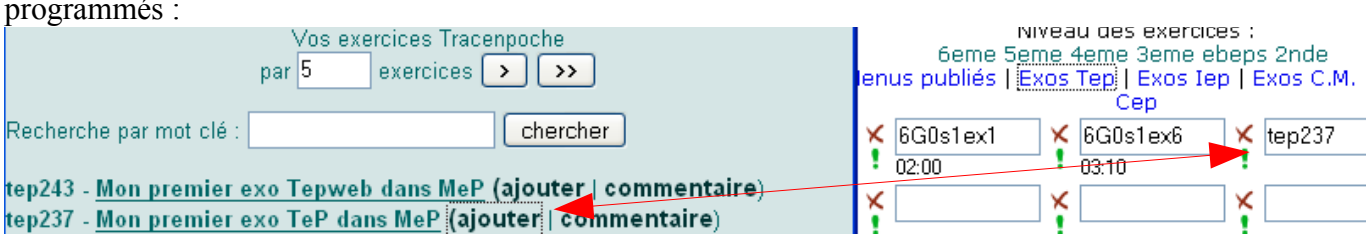


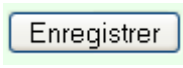
En cliquant sur l'un ou l'autre, une fenêtre s'ouvre affichant l'ensemble des exercices TeP que l'utilisateur a créés :



En cliquant sur "commentaire" on affiche les commentaires insérés lors de la création de l'exercice :

En cliquant sur "ajouter", on insère l'exercice TeP à sa séance MeP à la suite des exercices déjà programmés :



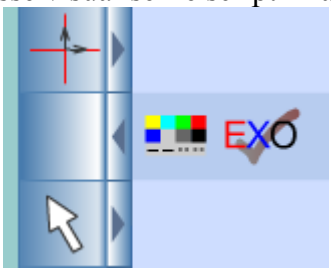
Il ne reste plus qu'à enregistrer sa séance : 

- **Déroulement de la séance.**

Lorsque l'élève se connecte à MeP et qu'une séance est programmée, il tombe sur la liste des exercices qui ont été choisis par l'enseignant :

Les exercices créés grâce à TeP sont différenciés de ceux de MeP.

Si l'exercice créé intègre l'interface TeP, l'élève doit appuyer sur le bouton **tepmep** à la fin de ses constructions pour que le professeur puisse visualiser le script final de la figure :

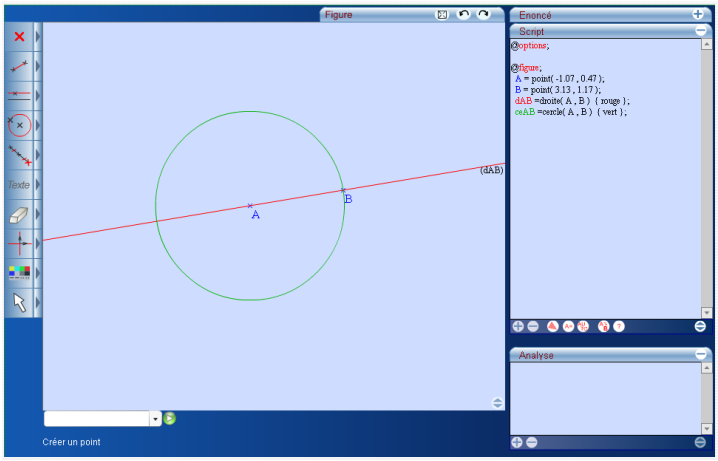


Par contre, si l'exercice intègre l'interface TepWeb, un bouton "Valider" est automatiquement intégré à la page sous la consigne. L'élève doit appuyer sur ce bouton avant de quitter l'exercice. Un message l'informera de la validation de son exercice.

- **Bilan de la séance.**

Lorsque l'on consulte le bilan d'une séance qui intègre des exercices créés avec TeP, la référence des exercices apparaît avec pour chacun la possibilité de "voir le travail" :

Si l'exercice intègre l'interface TeP, c'est la figure correspondant au script au moment où l'élève a appuyé sur le bouton  qui apparaît :



Il faut noter que la **zone script** sera toujours visible lors de la consultation, même si pour l'exercice elle avait été masquée.
Cela permet de vérifier autant la figure construite que la méthode de construction.

Si dans la consigne on a inséré une zone de texte, la réponse tapée par l'élève à l'intérieur apparaîtra sur la page "correction" :