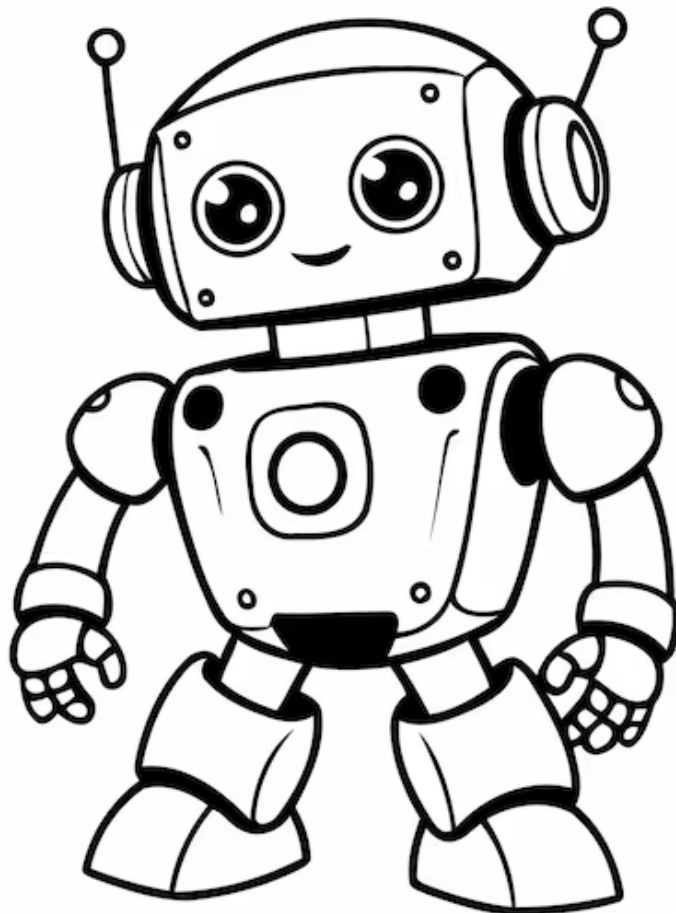


## DrawScript – Logiciel de dessin avec le langage Pascal

---

### Introduction

DrawScript est un logiciel développé par Sylvain Ard en 2025 pour dessiner avec le langage de programmation Pascal. Il est destiné aux enfants pour s'initier à la programmation. Un tutoriel sur le langage Pascal est joint.



## Chapitre 1 : Le canevas et les coordonnées X et Y

### Qu'est-ce que le canevas ?

Le **canevas**, c'est comme une feuille blanche sur laquelle tu vas dessiner. Dans DrawScript, le canevas est l'espace où ton dessin apparaîtra. Tu peux choisir la taille de cette "feuille" en définissant sa **largeur** et sa **hauteur** en pixels (un pixel est un tout petit point qui compose l'image).

---

### Étapes pour créer ton image

#### 1. Écrire ton code de dessin :

- Dans l'espace intitulé "**Code Pascal DrawScript**", tu vas écrire des instructions en langage Pascal pour dire à l'ordinateur quoi dessiner. Par exemple, pour dessiner une ligne, tu pourrais écrire une commande spécifique.

#### 2. Définir la taille de ton canevas :

- Juste en dessous de l'espace de code, tu verras deux champs :
  - "**Largeur de l'image (pixels)**" : Ici, tu indiques combien de pixels de large sera ton canevas.
  - "**Hauteur de l'image (pixels)**" : Ici, tu indiques combien de pixels de haut sera ton canevas.
- Par exemple, si tu veux une image carrée, tu peux mettre **400** en largeur et **400** en hauteur.

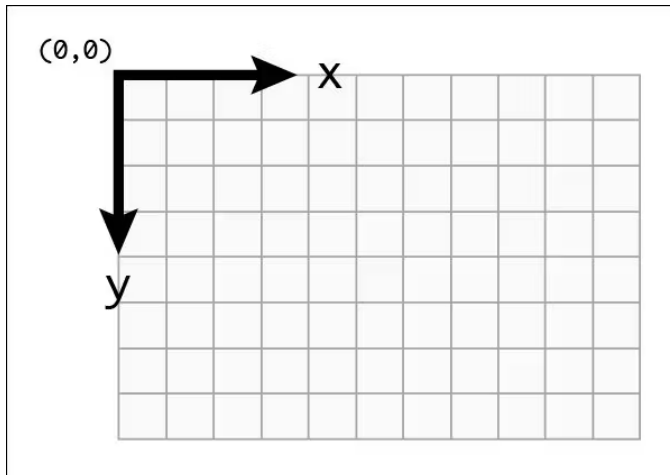
#### 3. Exécuter ton code pour voir le dessin :

- Une fois que tu as écrit ton code et défini la taille de ton canevas, clique sur le bouton "**Exécuter**".
  - Deux choses peuvent se passer :
    - **Si tout va bien** : Une nouvelle fenêtre s'ouvrira avec ton dessin affiché. Tu verras aussi un bouton "**Enregistrer au format PNG**" qui te permettra de sauvegarder ton image sur ton ordinateur.
    - **Si ton code contient des erreurs** : En bas de l'écran, une zone affichera des messages pour t'indiquer où se trouvent les erreurs dans ton code. Ne t'inquiète pas, c'est normal de faire des erreurs en apprenant. Lis les messages, corrige ton code, et essaie à nouveau !
-

## Amuse-toi bien !

DrawScript est un outil puissant qui te permet de créer des dessins en utilisant du code. Avec de la pratique, tu pourras réaliser des images de plus en plus complexes et impressionnantes. N'hésite pas à expérimenter et à t'amuser !

Voici le repère utilisé pour dessiner :



Tous les dessins que tu fais avec DrawScript se placent sur une grille invisible. Le coin en haut à gauche, c'est l'origine : le point (0, 0). Plus tu vas vers la droite, plus X augmente. Plus tu descends, plus Y augmente.











## Chapitre 2 : Les couleurs

Avant de commencer à dessiner, il faut choisir tes couleurs ! Avec la fonction `SetBrushColor`, tu peux colorier l'intérieur des formes que tu dessines. Il existe plein de couleurs prêtes à l'emploi comme `clRed`, `clBlue` ou `clGreen`.

Les couleurs en informatique sont souvent définies avec le modèle RVB. RVB veut dire Rouge, Vert et Bleu. Ce sont comme trois lampes de couleur qu'on peut allumer plus ou moins fort. Par exemple, si tu allumes seulement la lampe rouge au maximum (255) et les deux autres à zéro, tu obtiens du rouge pur. Si tu allumes le rouge et le vert à fond, mais pas le bleu, tu obtiens du jaune. Et si tu allumes les trois lampes à fond, tu obtiens du blanc ! Le Rouge (R), le bleu (B) et le vert (V) vont de 0 à 255.

Tu peux aussi créer ta propre couleur en utilisant la fonction `RGBColor`. Elle prend trois paramètres : R (rouge), V (vert) et B (bleu). Par exemple, `RGBColor(128, 0, 255)` donne une sorte de violet.

Tu peux aussi utiliser des codes couleur tous prêts, en voici la liste :

Code couleur	Nom français	Aperçu
clRed	Rouge	
clGreen	Vert	
clBlue	Bleu	
clYellow	Jaune	
clAqua	Cyan	
clFuchsia	Fuchsia	
clBlack	Noir	
clWhite	Blanc	
clGray	Gris	
clMaroon	Bordeaux	

Exemple avec RGBColor :

```
program MaCouleurPerso;
```

```
Begin
```

```
SetBrushColor(RGBColor(128, 0, 255));
```

```
FillRect(60, 60, 160, 160);
```

```
End.
```



Conseil du robot : Avec RGBColor, tu peux créer des milliers de couleurs différentes ! 🤖

### Chapitre 3 : Le pinceau






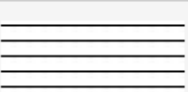


Le pinceau sert à colorier l'intérieur des formes. Tu peux choisir :

- Sa couleur avec SetBrushColor(couleur)
- Son style de remplissage avec SetBrushStyle(style)

La couleur est une constante comme `clRed`, ou une couleur personnalisée avec `RGBColor`.

Le style est un mot comme `bsSolid` ou `bsClear`.

Voici tous les styles de pinceau :

Valeur	Motif	Valeur	Motif
<code>bsSolid</code>		<code>bsCross</code>	
<code>bsClear</code>		<code>bsDiagCross</code>	
<code>bsBDiagonal</code>		<code>bsHorizontal</code>	
<code>bsFDiagonal</code>		<code>bsVertical</code>	

Voici un exemple de programme :

```
program MonPinceau;
```

```
Begin
```

```
    SetBrushColor (clBlue);
```

```
    SetBrushStyle (bsSolid);
```

```
    FillRect (50, 30, 130, 110);
```

```
    SetBrushColor (clWhite);
```

```
    SetBrushStyle (bsFDiagonal);
```

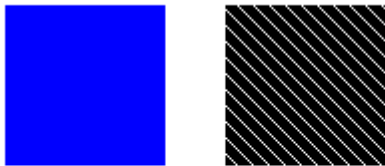
```
    FillRect (160, 30, 240, 110);
```

```
End.
```

Explications :

1. On commence le programme avec le mot-clé 'program'.
2. `SetBrushColor (clBlue)` ; choisit la couleur bleue.
3. `SetBrushStyle (bsSolid)` ; remplit le premier rectangle avec un style plein.
4. `FillRect (50, 30, 130, 110)` ; dessine un rectangle bleu.
5. `SetBrushColor (clBlue)` ; choisit la couleur blanche.
6. `SetBrushStyle (bsFDDiagonal)` ; change le style du pinceau.
7. `FillRect (160, 30, 240, 110)` ; dessine un autre rectangle avec des hachures.

Résultat attendu du code ci-dessus



Conseil du robot : Essaie tous les styles pour voir celui que tu préfères ! 🤖

## Chapitre 4 : Le stylo

Le **stylo** sert à **dessiner les contours** des formes.

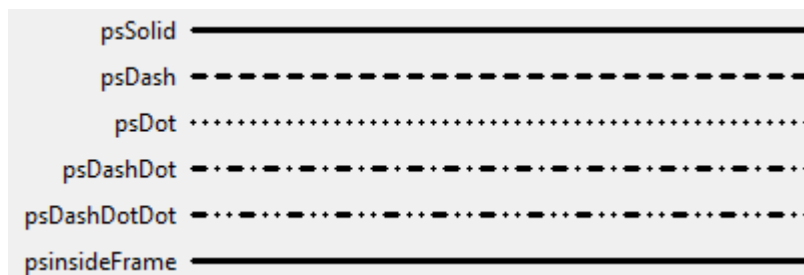
Quand tu fais un rectangle, un cercle ou une ligne, c'est le **stylo** qui décide de **la couleur**, de **l'épaisseur** et du **style** du trait.

Tu peux avoir un **trait noir très fin**, un **gros trait rouge**, ou même un **trait en pointillés**.  
C'est un peu comme choisir un vrai stylo ou un feutre avec un effet spécial !

Pour personnaliser ton stylo magique, tu peux utiliser plusieurs fonctions :

- **SetPenColor (couleur) :**  
Cette fonction change la **couleur** du trait.  
Par exemple, SetPenColor(clRed); donne un stylo rouge.
- **SetPenWidth (épaisseur) :**  
Elle permet de choisir si ton trait est **fin ou épais**.  
Un trait de largeur 1 est très fin, 5 est déjà bien gros !  
Exemple : SetPenWidth (3) ;
- **SetPenStyle (style) :**  
Cette fonction change le **style** du trait :  
tu peux faire un trait **continu, en pointillés, en tirets**, etc.  
Par exemple : SetPenStyle (psDash) ; pour un trait en tire

Voici tous les styles de stylos disponibles :



En plus il y a « psClear » qui est un stylo invisible, donc si tu dessines un rectangle avec ce stylo, tu n'en verras que l'intérieur avec le pinceau mais pas le contour.

Certains styles, comme psDash (tirets), psDot (pointillés) ou psDashDot, ne fonctionnent **correctement que si l'épaisseur du stylo est de 1**.

Si tu mets un trait plus épais, le style risque de ne **pas s'afficher comme prévu**.

Donc, si tu veux un **trait en pointillés**, pense à écrire :

```
SetPenWidth(1);
```

```
SetPenStyle(psDot);
```

## Exemple de programme

```
program MonStylo;  
  
Begin  
  
  SetPenColor(clBlue);  
  
  SetPenWidth(1);  
  
  SetPenStyle(psDash);  
  
  Rectangle(50, 50, 200, 120);  
  
End.
```

---

## Explication du code

- program MonStylo;  
C'est le nom de ton programme.
- Begin  
On commence le dessin.
- SetPenColor(clBlue);  
Le stylo est réglé sur la couleur **bleue**.
- SetPenWidth(1);  
Le trait aura une **épaisseur fine de 1 pixel de large**.
- SetPenStyle(psDash);  
Le contour du rectangle sera en **tirets**.
- Rectangle(50, 50, 200, 120);  
On dessine un **rectangle** avec ces coordonnées.
- End.  
Le dessin est terminé.



Résultat attendu :



## Chapitre 5 : Les polices

### ✂ Les fonctions pour écrire du texte

Voici toutes les fonctions que tu peux utiliser pour **changer l'apparence du texte** que tu écris sur ton image :

- **SetFontColor(couleur)**  
Cette fonction change la **couleur du texte**.  
Par exemple, SetFontColor(clRed); écrit ton texte en rouge.
- **SetFontSize(taille)**  
Elle permet de choisir la **taille du texte**.  
Une petite taille donne des lettres fines, une grande taille donne de grosses lettres.
- **SetFontStyle(bold, italic, underline)**  
Tu peux rendre ton texte **gras**, **italique**, ou **souligné**.  
Il faut écrire **true** (oui) ou **false** (non) pour chaque style.  
Par exemple :  
SetFontStyle(True, False, True); → gras et souligné.
- **SetFontFamily(nom)**  
Cette fonction change la **forme des lettres**, selon la police choisie.  
Tu peux écrire : SetFontFamily('Arial'); ou SetFontFamily('Times New Roman');  
(Attention à bien écrire le nom entre guillemets.)
- **TextWidth(texte)**  
Cette fonction te dit **quelle largeur occupera ton texte** sur l'image, en pixels.  
Exemple : TextWidth('Bonjour'); → retourne la largeur de ce mot.

- **TextHeight(texte)**  
Pareil que la largeur, mais pour la **hauteur du texte**.  
Très utile pour bien **positionner les textes** sur l'image.
- **TextRect(x1, y1, x2, y2, texte)**  
Cette fonction écrit un **texte à l'intérieur d'un rectangle**.  
Si le texte est trop long, il va **rester dans la boîte**.  
Pratique pour faire des **titres bien alignés** ou écrire dans des cases !
- **TextOut(x, y, texte)**  
Cette fonction permet d'**écrire du texte à un endroit précis** de l'image.  
Le texte commence au point (x, y) que tu choisis.  
Exemple : TextOut(100, 50, 'Coucou !'); écrit le mot "Coucou !" à 100 pixels en largeur et 50 pixels en hauteur.

### Exemple de programme

```
program MonTexte;

Begin

  SetFontColor(clGreen);

  SetFontSize(20);

  SetFontStyle(True, False, False);

  SetFontFamily('Arial');

  TextOut(50, 50, 'Bonjour !');

End.
```

---

### Explication du code

- program MonTexte;  
Le nom de ton programme.
- Begin  
On commence à dessiner sur l'image.
- SetFontColor(clGreen);  
Le texte sera écrit en **vert**.

- SetFontSize(20);  
La taille du texte est **assez grande**.
- SetFontStyle(True, False, False);  
Le texte sera en **gras**, mais **pas** en italique ni souligné.
- SetFontFamily('Arial');  
Les lettres seront écrites avec la police **Arial**.
- TextOut(50, 50, 'Bonjour !');  
On affiche le texte "**Bonjour !**" à la position (50, 50) sur l'image.
- End.





Résultat attendu :


**Bonjour !**

## Chapitre 6 : Le dessin

Maintenant que tu sais écrire du texte, tu vas apprendre à **dessiner des formes** sur ton image !

Tu peux créer plein de choses avec :

- des **rectangles** 
- des **cercles** et **ellipses** 
- des **lignes** 
- des **triangles** ou des **polygones** 

Grâce à ces formes de base, tu peux dessiner **des maisons, des bonshommes, des animaux, ou même des robots !** 

## Dessiner des lignes

Pour dessiner une **ligne droite** sur l'image, tu as deux façons de faire :

### Méthode 1 : avec MoveTo et LineTo

- **MoveTo(x, y)**  
Cette commande **place le stylo à un point**, sans rien dessiner.  
C'est comme si tu posais ton crayon sur la feuille sans appuyer.
- **LineTo(x, y)**  
Cette commande **trace une ligne depuis la position actuelle** du stylo jusqu'au point (x, y).  
C'est comme si tu faisais un trait avec ta règle !

Exemple :

```
MoveTo(50, 100);
```

```
LineTo(200, 100);
```

Cela trace une ligne de gauche à droite.

### Méthode 2 : avec Line(x1, y1, x2, y2)

- **Line(x1, y1, x2, y2)**  
Cette commande **trace directement une ligne** entre deux points : (x1, y1) et (x2, y2).  
C'est plus rapide si tu veux juste une seule ligne.

Exemple :

```
Line(50, 150, 200, 150);
```

→ Cela trace une autre ligne, un peu plus bas.

### Exemple : Un triangle avec MoveTo et LineTo

```
program MonTriangle;
```

```
Begin
```

```
MoveTo(100, 50);
```

```
LineTo(50, 150);
```

```
LineTo(150, 150);
```

```
LineTo(100, 50);
```

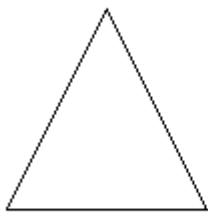
End.

---

### Explication du code

- `program MonTriangle;`  
Le nom du programme.
- `Begin`  
On commence le dessin.
- `MoveTo(100, 50);`  
On place le stylo tout en haut du triangle (le sommet).
- `LineTo(50, 150);`  
On trace une ligne vers le coin inférieur gauche.
- `LineTo(150, 150);`  
Puis une ligne vers le coin inférieur droit.
- `LineTo(100, 50);`  
Et enfin, on revient au sommet pour fermer le triangle.
- `End.`  
Fin du programme.

Résultat attendu :



## Les rectangles

Tu peux dessiner plusieurs sortes de **rectangles** avec ces fonctions :

- **Rectangle(x1, y1, x2, y2)**  
Dessine un rectangle **classique**, avec des coins droits.  
(x1, y1) est le coin **en haut à gauche**, et (x2, y2) est le coin en bas à droite.
- **RoundRect(x1, y1, x2, y2, rx, ry)**  
Dessine un rectangle avec **les coins arrondis** !  
rx et ry indiquent de **combien de pixels les coins sont arrondis** en largeur (rx) et en hauteur (ry).  
  
(x1, y1) est le coin **en haut à gauche**, et (x2, y2) est le coin en bas à droite.
- **FillRect(x1, y1, x2, y2)**  
Remplit un rectangle **sans dessiner de contour**.  
La couleur utilisée est celle du **pinceau** (SetBrushColor).  
C'est très pratique pour faire un fond coloré.  
  
(x1, y1) est le coin **en haut à gauche**, et (x2, y2) est le coin en bas à droite.

### Exemple : Trois rectangles différents

```
program TroisRectangles;  
  
Begin  
  
  // Rectangle classique rouge  
  
  SetPenColor(clRed);  
  
  SetBrushColor(clWhite);  
  
  Rectangle(50, 50, 150, 120);  
  
  // Rectangle arrondi vert  
  
  SetPenColor(clGreen);  
  
  SetBrushColor(clWhite);  
  
  RoundRect(200, 50, 350, 120, 20, 20);  
  
  // Rectangle rempli bleu (sans contour)
```

```
SetBrushColor(clBlue);
```

```
FillRect(100, 200, 300, 270);
```

End.

### Explication du code

- program TroisRectangles;  
Le nom du programme.
  - Begin  
Début du dessin.
- 

### 1er rectangle : classique

- SetPenColor(clRed);  
Le contour sera rouge.
  - SetBrushColor(clWhite);  
L'intérieur du rectangle sera blanc.
  - Rectangle(50, 50, 150, 120);  
On dessine un rectangle avec des coins droits, en haut à gauche de l'image.
- 

### 2e rectangle : arrondi

- SetPenColor(clGreen);  
Le contour sera vert.
- SetBrushColor(clWhite);  
L'intérieur sera blanc aussi.
- RoundRect(200, 50, 350, 120, 20, 20);  
On dessine un rectangle avec des coins arrondis (20 pixels de rayon).

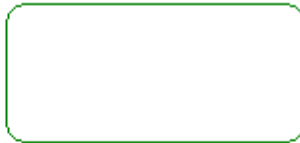
---

### ■ 3e rectangle : rempli sans contour

- `SetBrushColor(clBlue);`  
L'intérieur sera bleu.
- `FillRect(100, 200, 300, 270);`  
Ce rectangle **n'a pas de contour**, il est juste **plein**.  
Il est placé plus bas pour ne pas chevaucher les autres.

- 
- `End.`  
Fin du dessin.

Résultat attendu :



### ◆ Dessiner un polygone

Un **polygone**, c'est une forme avec **plusieurs côtés** : triangle, losange, pentagone...

Avec DrawScript, tu peux dessiner un polygone **en plusieurs étapes**, en donnant **chaque point un par un**.



Voici les trois fonctions à utiliser :

- **BeginPolygon(x, y)**  
Tu commences un nouveau polygone et tu indiques **le premier point**.
- **ContinuePolygon(x, y)**  
Tu ajoutes un nouveau **point à la suite**.  
Chaque fois que tu appelles cette fonction, un **nouveau côté** est dessiné.
- **EndPolygon**  
Tu **termine le polygone** : la dernière ligne relie le dernier point au premier, pour **fermer la forme**.

💡 Astuce : entre les appels, tu peux changer la **couleur du stylo**, **l'épaisseur**, ou encore la couleur de **remplissage** !

🖋 **Exemple : Un losange avec BeginPolygon et cie**

```
program MonPolygone;
```

```
Begin
```

```
SetPenColor(clNavy);
```

```
SetBrushColor(clAqua);
```

```
BeginPolygon(200, 100);
```

```
ContinuePolygon(300, 200);
```

```
ContinuePolygon(200, 300);
```

```
ContinuePolygon(100, 200);
```

```
EndPolygon;
```

```
End.
```

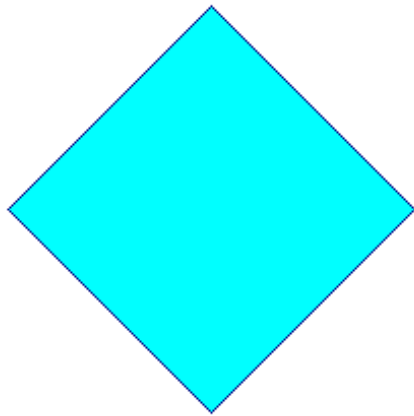
## Explication du code

- `program MonPolygone;`  
Le nom du programme.
  - `Begin`  
On commence le dessin sur l'image.
  - `SetPenColor(clNavy);`  
Le contour du polygone sera en **bleu foncé**.
  - `SetBrushColor(clAqua);`  
L'intérieur du polygone sera **bleu clair**.
- 

## Création du polygone (losange)

- `BeginPolygon(200, 100);`  
On place le **premier point** du losange, tout en haut.
  - `ContinuePolygon(300, 200);`  
Deuxième point : **à droite**.
  - `ContinuePolygon(200, 300);`  
Troisième point : **en bas**.
  - `ContinuePolygon(100, 200);`  
Quatrième point : **à gauche**.
  - `EndPolygon;`  
Le dernier point est relié au premier, et le **losange est fermé** et rempli.
- 
- `End.`  
Fin du programme.

Résultat attendu :



## 🕒 Les formes arrondies : ellipses et arcs

Une **ellipse**, c'est comme un **cercle allongé** : parfois un peu plus large, parfois plus haut, un peu comme un **œuf**.

On peut en dessiner **entièrement** ou juste **une partie** (comme une tranche de pizza 🍕).

Voici les fonctions pour dessiner des **formes rondes** :

---

### 1. Ellipse(x1, y1, x2, y2)

Cette fonction dessine une **ellipse complète**, ou un cercle si les côtés sont égaux.

- (x1, y1) : coin **en haut à gauche** du rectangle qui contient l'ellipse.
- (x2, y2) : coin **en bas à droite** du rectangle.

💡 Imagine que tu dessines un **œuf dans un cadre** : (x1, y1) et (x2, y2) définissent ce cadre.

---

## 2. Arc(x1, y1, x2, y2, x3, y3, x4, y4)

Cette fonction dessine **un morceau de l'ellipse**, un **arc** comme un **bout d'arc-en-ciel**.

- (x1, y1) et (x2, y2) : définissent l'ellipse complète (le cadre, comme pour Ellipse).
- (x3, y3) : point de **départ** de l'arc, **sur le bord** de l'ellipse.
- (x4, y4) : point de **fin** de l'arc, **aussi sur le bord**.

💡 Tu dis : « Je veux **partir de là**, et **aller jusque là**, en suivant le bord de l'ellipse ».

---

## 3. Chord(x1, y1, x2, y2, x3, y3, x4, y4)

Un **chord** (corde) est **comme un arc**, mais **fermé par une ligne droite**.

- Même chose que Arc pour tous les points.
- Mais ici, on **trace un trait droit** entre les deux extrémités, ce qui fait **une forme fermée**.

💡 Ça ressemble à **un petit morceau coupé dans une ellipse**, comme si on avait fait une découpe au couteau.

---

## 4. Pie(x1, y1, x2, y2, x3, y3, x4, y4)

Un **pie** (tarte) dessine aussi une **partie d'ellipse**, mais cette fois, on **relie les extrémités au centre** de l'ellipse.

- (x3, y3) : point de **départ** sur le bord de l'ellipse.
- (x4, y4) : point de **fin** sur le bord aussi.
- On relie ces deux points au **centre de l'ellipse** pour faire une **tranche de tarte** !

💡 Imagine une **part de camembert** : un bout de cercle, avec deux lignes qui vont au centre.

Dans les fonctions Arc, Chord et Pie, les points (x3, y3) et (x4, y4) doivent être **sur la bordure de l'ellipse** (c'est-à-dire sur sa **circonférence**).

👉 Ces deux points servent à indiquer :

- **où commence** l'arc (le point de départ),
- et **où il se termine** (le point d'arrivée),

### 🧠 Astuce du robot : comment trouver un point sur le bord d'une ellipse

Imaginons que tu as une ellipse dans un cadre défini par (x1, y1) et (x2, y2).

Tu veux trouver un point **sur le bord** de cette ellipse, à un certain **angle** (comme les aiguilles d'une montre 🕒).

Voici comment faire :

```
angle := 45; // en degrés
```

```
xc := (x1 + x2) div 2; // centre X
```

```
yc := (y1 + y2) div 2; // centre Y
```

```
rx := (x2 - x1) div 2; // rayon horizontal
```

```
ry := (y2 - y1) div 2; // rayon vertical
```

```
x := Round(xc + rx * cos(angle * Pi / 180));
```

```
y := Round(yc - ry * sin(angle * Pi / 180));
```

### 🌸 Explications simples :

- xc et yc sont les **coordonnées du centre** de l'ellipse.
- rx et ry sont les **rayons** (demi-largeur, demi-hauteur).
- angle est l'angle que tu choisis en **degrés** (0° = droite, 90° = haut, 180° = gauche, 270° = bas).
- La formule cos(...) et sin(...) permet de **trouver un point sur la bordure**, à cet angle précis.

💡 Tu peux donc facilement trouver les points (x3, y3) et (x4, y4) en choisissant deux angles différents !

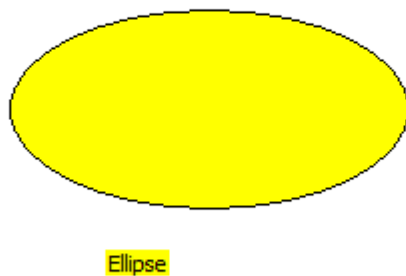
Round est une fonction mathématique pour arrondir un nombre flottant en entier, cos est le cosinus et sin le sinus.

Exemple de code :

### **Exemple 1 : Ellipse**

```
program ExempleEllipse;  
  
Begin  
  
  SetPenColor(clBlack);  
  
  SetBrushColor(clYellow);  
  
  Ellipse(100, 100, 300, 200);  
  
  TextOut(150, 220, 'Ellipse');  
  
End.
```

Résultat :



### **Exemple 2 : Arc**

```
program ExempleArc;  
  
Var  
  
  xc, yc, rx, ry: Integer;  
  
  x1, y1, x2, y2: Integer;  
  
  px1, py1, px2, py2: Integer;
```

Begin

```
x1 := 100; y1 := 100; x2 := 300; y2 := 200;
```

```
xc := (x1 + x2) div 2;
```

```
yc := (y1 + y2) div 2;
```

```
rx := (x2 - x1) div 2;
```

```
ry := (y2 - y1) div 2;
```

```
// Bien respecter les rayons pour rester sur la bordure de l'ellipse
```

```
px1 := xc + Round(rx * cos(Pi / 4)); // 45°
```

```
py1 := yc - Round(ry * sin(Pi / 4));
```

```
px2 := xc + Round(rx * cos(3 * Pi / 4)); // 135°
```

```
py2 := yc - Round(ry * sin(3 * Pi / 4));
```

```
SetPenColor(clRed);
```

```
Arc(x1, y1, x2, y2, px1, py1, px2, py2);
```

```
TextOut(160, 220, 'Arc');
```

End.

Résultat :



Arc

### Exemple 3 : Chord

```
program ExempleChord;
```

```
Var
```

```
xc, yc, rx, ry: Integer;
```

```
x1, y1, x2, y2, px1, py1, px2, py2: Integer;
```

```
Begin
```

```
x1 := 50; y1 := 50; x2 := 350; y2 := 200;
```

```
xc := (x1 + x2) div 2;
```

```
yc := (y1 + y2) div 2;
```

```
rx := (x2 - x1) div 2;
```

```
ry := (y2 - y1) div 2;
```

```
px1 := Round(xc + rx * cos(30 * Pi / 180));
```

```
py1 := Round(yc - ry * sin(30 * Pi / 180));
```

```
px2 := Round(xc + rx * cos(150 * Pi / 180));
```



```
py2 := Round(yc - ry * sin(150 * Pi / 180));
```

```
SetBrushColor(clAqua);
```

```
SetPenColor(clTeal);
```

```
Chord(x1, y1, x2, y2, px1, py1, px2, py2);
```

```
TextOut(160, 220, 'Chord');
```

End.

Résultat :



Chord

#### **Exemple 4 : Pie**

```
program ExemplePie;
```

```
Var
```

```
xc, yc, rx, ry: Integer;
```

```
x1, y1, x2, y2, px1, py1, px2, py2: Integer;
```

```
Begin
```

```
x1 := 50; y1 := 50; x2 := 350; y2 := 200;
```

```
xc := (x1 + x2) div 2;
```

```
yc := (y1 + y2) div 2;
```

```
rx := (x2 - x1) div 2;
```

```
ry := (y2 - y1) div 2;
```

```
px1 := Round(xc + rx * cos(60 * Pi / 180));
```

```
py1 := Round(yc - ry * sin(60 * Pi / 180));
```

```
px2 := Round(xc + rx * cos(120 * Pi / 180));
```

```
py2 := Round(yc - ry * sin(120 * Pi / 180));
```

```
SetBrushColor(clFuchsia);
```

```
SetPenColor(clPurple);
```

```
Pie(x1, y1, x2, y2, px1, py1, px2, py2);
```

```
TextOut(170, 220, 'Pie');
```

End.

Résultat :

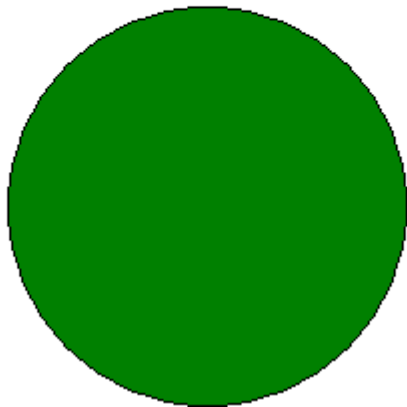


Pie

### Exemple 5 : Cercle (via Ellipse)

```
program ExempleCercle;  
  
Begin  
  
  SetPenColor(clBlack);  
  
  SetBrushColor(clGreen);  
  
  Ellipse(100, 100, 300, 300); // largeur = hauteur → cercle parfait  
  
  TextOut(160, 320, 'Cercle');  
  
End.
```

Résultat :



### Qu'est-ce qu'une courbe de Bézier ?

Imagine que tu veux dessiner une **belle courbe lisse** qui passe par certains endroits précis. Les **courbes de Bézier** sont des outils mathématiques qui permettent de créer de telles courbes en utilisant quelques **points de contrôle**. Ces courbes sont très utilisées dans les logiciels de dessin et d'animation pour créer des formes douces et naturelles.

---

## Les points de contrôle

Une courbe de Bézier est définie par plusieurs **points de contrôle**. Prenons l'exemple d'une **courbe de Bézier cubique**, qui utilise **quatre points de contrôle** :

1. **Point de départ** : C'est le point où la courbe commence.
2. **Premier point de contrôle intermédiaire** : Il "attire" la courbe au début, déterminant la direction initiale.
3. **Deuxième point de contrôle intermédiaire** : Il "attire" la courbe vers la fin, influençant sa trajectoire finale.
4. **Point d'arrivée** : C'est le point où la courbe se termine.

La courbe commence au **point de départ**, est "attirée" vers le **premier point de contrôle intermédiaire**, puis vers le **deuxième point de contrôle intermédiaire**, et enfin atteint le **point d'arrivée**. Cependant, la courbe ne passe généralement pas par les points de contrôle intermédiaires ; ces points servent plutôt de "guides" pour façonner la courbe.

---

## Dessiner une courbe de Bézier avec DrawScript

Dans DrawScript, tu peux dessiner des courbes de Bézier en utilisant trois fonctions principales :

1. **BeginBezier(x1, y1, x2, y2, x3, y3, x4, y4)**  
Cette fonction commence une nouvelle courbe de Bézier cubique.
  - (x1, y1) : Coordonnées du point de départ.
  - (x2, y2) : Coordonnées du premier point de contrôle intermédiaire.
  - (x3, y3) : Coordonnées du deuxième point de contrôle intermédiaire.
  - (x4, y4) : Coordonnées du point d'arrivée.
2. **ContinueBezier(x1, y1, x2, y2, x3, y3)**  
Cette fonction prolonge la courbe de Bézier en ajoutant une nouvelle section qui commence là où la précédente s'est arrêtée.
  - (x1, y1) : Coordonnées du premier point de contrôle intermédiaire pour la nouvelle section.
  - (x2, y2) : Coordonnées du deuxième point de contrôle intermédiaire pour la nouvelle section.
  - (x3, y3) : Coordonnées du nouveau point d'arrivée.

### 3. **EndBezier()**

Cette fonction termine la courbe de Bézier en cours.

Exemple de programme :

```
program CourbeBezierComplexe;
```

```
Begin
```

```
  SetPenColor(clBlue);
```

```
  // Premier segment de Bézier
```

```
  BeginBezier(50, 150, 150, 50, 250, 250, 350, 150);
```

```
  // Deuxième segment de Bézier
```

```
  ContinueBezier(450, 50, 550, 250, 650, 150);
```

```
  // Troisième segment de Bézier
```

```
  ContinueBezier(750, 50, 850, 250, 950, 150);
```

```
  // Terminer la courbe de Bézier
```

```
  EndBezier;
```

```
End.
```

Dans cet exemple :

La courbe commence au point (50, 150).

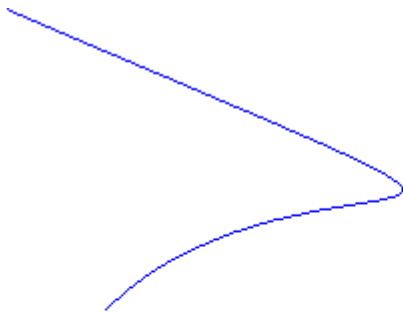
Le premier segment est défini par les points de contrôle (150, 50), (250, 250) et se termine au point (350, 150).

Le deuxième segment continue depuis le point précédent, avec des points de contrôle (450, 50), (550, 250) et se termine au point (650, 150).

Le troisième segment poursuit avec des points de contrôle (750, 50), (850, 250) et se termine au point (950, 150).

EndBezier termine la séquence de courbes de Bézier.

Résultat :



N'hésites pas à modifier les coordonnées des points pour observer comment la courbe change en fonction des points de contrôle.

### Qu'est-ce que la fonction SetPixel ?

La fonction SetPixel permet de **colorer un point précis** (appelé **pixel**) sur une image à une position donnée. C'est comme si tu coloriais un seul petit carré sur une grande feuille de papier quadrillée.

---

## **Comment utiliser SetPixel dans DrawScript ?**

Dans DrawScript, la fonction SetPixel s'utilise ainsi :

SetPixel(x, y, color);

- x : Position horizontale du pixel (distance depuis le bord gauche).
- y : Position verticale du pixel (distance depuis le haut).
- color : Couleur à appliquer au pixel.

La couleur est généralement spécifiée en utilisant des constantes prédéfinies comme clRed pour rouge, clBlue pour bleu, etc.

---

## **Exemple simple**

Voici un petit programme qui colore un pixel en rouge au centre d'une image de 400x400 pixels :

```
program DessinerPixel;
```

```
Begin
```

```
    SetPixel(200, 200, clRed);
```

```
End.
```

Dans cet exemple, le pixel situé aux coordonnées (200, 200) sera coloré en rouge.

---

## **Exemple avancé : Dessiner une ligne point par point**

En répétant l'utilisation de SetPixel, on peut dessiner des formes plus complexes, comme une ligne diagonale :

```
program LigneDiagonale;
```

```
Var
```

```
    i: Integer;
```

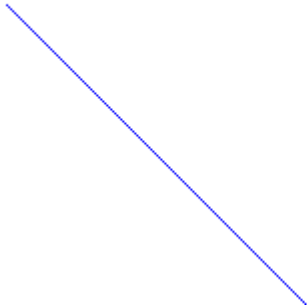
```
Begin
```

```
    For i := 50 To 350 Do
```

```
        SetPixel(i, i, clBlue);
```

```
End.
```

Résultat :



Ce programme dessine une ligne bleue allant du point (50, 50) au point (350, 350).

---

### Remarque importante

L'utilisation intensive de SetPixel peut ralentir le dessin si tu dessines beaucoup de pixels individuellement. Pour des formes complexes, il est souvent préférable d'utiliser des fonctions de dessin plus avancées fournies par DrawScript, comme celles pour dessiner des lignes, des rectangles ou des cercles.

---

En résumé, la fonction SetPixel est un outil simple mais puissant pour manipuler individuellement les pixels d'une image dans DrawScript. Elle te permet de contrôler précisément l'apparence de chaque point de ton dessin.

### Qu'est-ce que la fonction FloodFill ?

La fonction FloodFill te permet de **remplir une zone entière** d'une image avec une couleur spécifique. C'est comme utiliser un **pot de peinture** dans un logiciel de dessin : en cliquant sur une zone, toute la région connectée est remplie de la couleur choisie.

---



## Comment utiliser FloodFill dans DrawScript ?

Dans **DrawScript**, la fonction FloodFill s'utilise ainsi :

FloodFill(x, y, color);

- x : Coordonnée horizontale du point de départ du remplissage.
- y : Coordonnée verticale du point de départ du remplissage.
- color : Couleur utilisée pour le remplissage.

Avant d'appeler FloodFill, il est important de définir la couleur de remplissage en utilisant la fonction SetBrushColor.

---

## Exemple simple

Voici un exemple de programme qui dessine un cercle puis remplit son intérieur avec une couleur spécifique :

```
program ExempleFloodFill;
```

```
Begin
```

```
// Définir la couleur du stylo (contour)
```

```
SetPenColor(clBlack);
```

```
// Dessiner un cercle avec centre en (200, 200) et rayon de 100 pixels
```

```
Ellipse(100, 100, 300, 300);
```

```
// Définir la couleur de remplissage
```

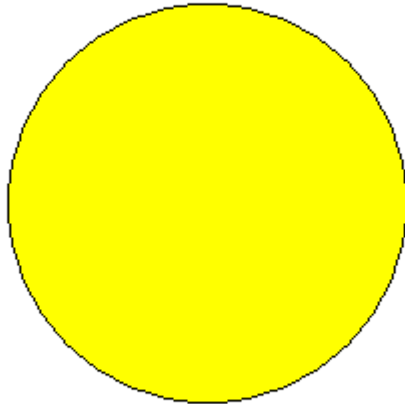
```
SetBrushColor(clYellow);
```

```
// Remplir l'intérieur du cercle en partant du point central
```

```
FloodFill(200, 200, clBlack);
```

```
End.
```

Résultat :



#### Explications :

1. **Définition de la couleur du contour** : `SetPenColor(clBlack)`; définit la couleur du contour en noir.
2. **Dessin du cercle** : `Ellipse(100, 100, 300, 300)`; dessine un cercle avec un rectangle englobant défini par les coins supérieurs gauche (100, 100) et inférieurs droit (300, 300).
3. **Définition de la couleur de remplissage** : `SetBrushColor(clYellow)`; définit la couleur de remplissage en jaune.
4. **Remplissage de l'intérieur du cercle** : `FloodFill(200, 200, clBlack)`; remplit la zone entourée par la couleur noire en partant du point (200, 200).

---

#### Remarques importantes

- **Point de départ valide** : Assure-toi que le point de départ (x, y) se trouve à l'intérieur de la zone que tu souhaites remplir et que cette zone est entièrement entourée par une bordure d'une couleur différente pour éviter que le remplissage ne déborde.

- **Définition préalable du contour** : Si la zone à remplir n'a pas de contour défini, FloodFill peut remplir l'ensemble de l'image.
- **Performance** : L'utilisation de FloodFill sur de grandes zones peut être gourmande en ressources. Il est recommandé de l'utiliser sur des zones de taille modérée pour maintenir de bonnes performances.

---

En résumé, la fonction FloodFill de **DrawScript** est un outil puissant pour remplir des zones délimitées dans tes dessins. En maîtrisant son utilisation, tu pourras créer des illustrations colorées et détaillées avec précision.

### **Fonction Draw**

La fonction Draw est utilisée pour dessiner une image à une position spécifique sur une autre image ou un canevas. Elle place l'image source sans modifier sa taille.

#### **Syntaxe :**

Draw(x, y, Image);

- x, y : Coordonnées du point supérieur gauche où l'image sera dessinée.
- Image : L'image que tu souhaites dessiner.

#### **Exemple :**

program ExempleDraw;

Begin

// Charger l'image source

var Image := LoadImage('chemin/vers/image.png');

// Dessiner l'image aux coordonnées (50, 100)

Draw(50, 100, Image);

End.

Dans cet exemple, l'image chargée est dessinée à la position (50, 100) sur le canevas actuel sans modification de sa taille.

---

## **Fonction StretchDraw**

La fonction StretchDraw permet de dessiner une image en l'adaptant à une zone rectangulaire spécifiée, ce qui peut entraîner un redimensionnement de l'image source pour qu'elle s'ajuste aux dimensions du rectangle cible.

### **Syntaxe :**

StretchDraw(x1, y1, x2, y2, Image);

- x1, y1 : Coordonnées du coin supérieur gauche du rectangle cible.
- x2, y2 : Coordonnées du coin inférieur droit du rectangle cible.
- Image : L'image que tu souhaites dessiner et redimensionner.

### **Exemple :**

program ExempleStretchDraw;

Begin

// Charger l'image source

var Image := LoadImage('chemin/vers/image.png');

// Définir le rectangle cible

var x1 := 50;

var y1 := 100;

var x2 := 200;

var y2 := 300;

// Dessiner l'image en l'adaptant au rectangle spécifié

StretchDraw(x1, y1, x2, y2, Image);

End.

Dans cet exemple, l'image chargée est dessinée dans le rectangle défini par les coins (50, 100) et (200, 300). Si l'image source a des dimensions différentes, elle sera redimensionnée pour s'adapter à ce rectangle, ce qui peut entraîner une distorsion si les proportions ne sont pas conservées.

---

## Remarques Importantes

- **Qualité de l'image** : Lors de l'utilisation de StretchDraw, le redimensionnement peut entraîner une perte de qualité ou une distorsion de l'image si les proportions ne sont pas maintenues. Pour préserver les proportions, assure-toi que le rapport largeur/hauteur du rectangle cible correspond à celui de l'image source. De plus, l'utilisation de méthodes de redimensionnement avec interpolation peut améliorer la qualité du rendu.
- **Performances** : Le redimensionnement d'images peut être gourmand en ressources, surtout pour de grandes images. Il est recommandé d'utiliser des images de tailles appropriées pour minimiser le besoin de redimensionnement et optimiser les performances.

---

En résumé, les fonctions Draw et StretchDraw de **DrawScript** te permettent de dessiner des images sur un canevas, avec ou sans redimensionnement. En comprenant leur fonctionnement et en prenant en compte les considérations de qualité et de performance, tu pourras intégrer efficacement des images dans tes projets de dessin.

## Qu'est-ce que la fonction GradientFill ?

La fonction GradientFill te permet de **remplir une zone rectangulaire** avec un **dégradé de couleurs**, créant ainsi une transition douce entre deux couleurs. C'est similaire à mélanger deux peintures sur une feuille pour obtenir un effet progressif.

---

## Comment utiliser GradientFill ?

Voici comment est définie la fonction GradientFill :

```
procedure GradientFill(AStartColor, AEndColor: Integer; x, y, width, height: Integer; DirectionPS: Integer);
```

- AStartColor : La couleur de départ du dégradé.[YouTube](#)
- AEndColor : La couleur d'arrivée du dégradé.[YouTube](#)
- x, y : Les coordonnées du coin supérieur gauche du rectangle à remplir.
- width, height : La largeur et la hauteur du rectangle.
- DirectionPS : La direction du dégradé.

Pour définir la direction du dégradé, tu as créé deux constantes :

- gdHorizontal : Valeur 1, pour un dégradé **horizontal** (de gauche à droite).

- gdVertical : Valeur 2, pour un dégradé **vertical** (de haut en bas).

---

### 🧠 Exemple d'utilisation de GradientFill

Imaginons que tu veuilles remplir un rectangle de 200 pixels de large et 100 pixels de haut, situé à la position (50, 50), avec un dégradé allant du rouge au bleu, dans une direction horizontale. Voici comment tu peux le faire :

Program Degrade ;

Begin

```
// Remplir un rectangle avec un dégradé horizontal du rouge au bleu
```

```
GradientFill(clRed, clBlue, 50, 50, 200, 100, gdHorizontal);
```

End.

Résultat :



Dans cet exemple :

- clRed et clBlue sont des constantes représentant les couleurs rouge et bleu.
- 50, 50 sont les coordonnées du coin supérieur gauche du rectangle.
- 200, 100 sont la largeur et la hauteur du rectangle.
- gdHorizontal indique que le dégradé doit s'effectuer de gauche à droite.

---

### 🔪 Expérimente avec les dégradés !

Tu peux t'amuser à modifier les couleurs, les dimensions et la direction pour voir comment cela affecte le dégradé. Par exemple, pour un dégradé vertical du vert au jaune :

Program Degrade ;

Begin

```
// Remplir un rectangle avec un dégradé vertical du vert au jaune
```

```
GradientFill(clGreen, clYellow, 100, 100, 150, 150, gdVertical);
```

End.

Résultat :



---

En utilisant la fonction GradientFill, tu peux ajouter de magnifiques effets de dégradé à tes dessins, rendant tes créations encore plus attrayantes et colorées !



### Qu'est-ce que la fonction DrawArrow ?

La fonction DrawArrow te permet de **dessiner une flèche** entre deux points spécifiques sur ton image. C'est comme tracer une ligne entre deux endroits et ajouter une pointe de flèche à l'une des extrémités pour indiquer une direction.



### Comment utiliser DrawArrow ?

Voici comment est définie la fonction DrawArrow :

```
procedure DrawArrow(xx1, yy1, xx2, yy2: Integer);
```

- xx1, yy1 : Coordonnées du **point de départ** de la flèche.
- xx2, yy2 : Coordonnées du **point d'arrivée** de la flèche (là où se trouve la pointe).

## 🧠 Exemple d'utilisation de DrawArrow

Imaginons que tu veuilles dessiner une flèche partant du point (100, 150) et pointant vers le point (300, 150). Voici comment tu peux le faire :

Program Fleche ;

Begin

```
// Dessiner une flèche de (100, 150) à (300, 150)
```

```
DrawArrow(100, 150, 300, 150);
```

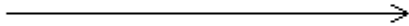
End.

Dans cet exemple :

- La flèche commence au point (100, 150).
- Elle pointe vers le point (300, 150).

---

Résultat :



## 📏 Dessiner en fonction de la taille du canevas

Parfois, tu veux que ton dessin **s'adapte automatiquement à la taille de l'image**, peu importe si elle fait 300x200 ou 800x600.

C'est là que les fonctions **CanvasWidth** et **CanvasHeight** sont très utiles !

---



### 🧠 Que font CanvasWidth et CanvasHeight ?

- **CanvasWidth** : donne la **largeur** actuelle du canevas (en pixels).
- **CanvasHeight** : donne la **hauteur** actuelle du canevas (en pixels).

Tu peux les utiliser dans ton code comme des **valeurs dynamiques**. Par exemple :

```
Line(0, 0, CanvasWidth, CanvasHeight);
```

→ Cette ligne commence dans le coin en haut à gauche et va jusqu'en bas à droite, **quelle que soit la taille du canevas**.

### 🔧 Exemple simple avec CanvasWidth et CanvasHeight

```
program CroixDiagonale;
```

```
Begin
```

```
SetPenColor(clRed);
```

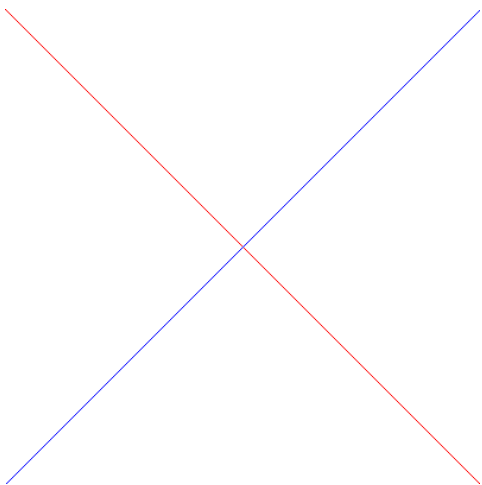
```
Line(0, 0, CanvasWidth, CanvasHeight);
```

```
SetPenColor(clBlue);
```

```
Line(CanvasWidth, 0, 0, CanvasHeight);
```

```
End.
```

Résultat :



## Conclusion

Félicitations pour avoir complété ce tutoriel sur **DrawScript** ! Au fil des chapitres, tu as exploré une variété de fonctions, depuis les outils de dessin jusqu'aux fonctions mathématiques avancées, te permettant de créer des images dynamiques et proportionnelles.

En maîtrisant des fonctions telles que `CanvasWidth` et `CanvasHeight`, tu es désormais capable de concevoir des dessins qui s'adaptent automatiquement à la taille du canevas, rendant tes créations plus flexibles et professionnelles.

N'oublie pas que la programmation et le dessin sont des compétences qui s'affinent avec la pratique. Continue à expérimenter, à poser des questions et à explorer de nouvelles idées. Chaque ligne de code que tu écris te rapproche de la maîtrise de **DrawScript** et enrichit ta compréhension de la programmation graphique.

Si tu rencontres des défis ou souhaites approfondir certains concepts, n'hésite pas à revenir sur les sections précédentes du tutoriel ou à rechercher des ressources supplémentaires. La communauté des programmeurs est vaste et toujours prête à partager des connaissances.

Encore bravo pour ton engagement et ta persévérance. Que cette aventure avec **DrawScript** soit le début d'un passionnant voyage dans le monde de la programmation et de la création graphique !