

Tutoriel langage Pascal pour débutants

Introduction

Sylvain Ard en 2025 a écrit deux logiciels pour que les collégiens s'initient à la programmation :

- RadioScript qui permet de jouer des fichiers musicaux.
- DrawScript qui permet de dessiner.

Ces deux logiciels sont dans le langage Pascal.

Ce tutoriel commence par introduire les interfaces communes, puis introduit le langage Pascal, puis introduit les fonctions communes à RadioScript et DrawScript puis introduit la seule fonction spécifique à RadioScript. Un tutoriel complémentaire sur les spécificités de DrawScript est disponible dans son ZIP.

Interfaces de RadioScript et DrawScript

Ecrivez votre code dans la fenêtre de gauche (pour RadioScript) ou dans la grande boîte (pour DrawScript) puis cliquez sur « Exécuter », la fenêtre de droite (pour RadioScript) ou la boîte de bas (pour DrawScript) écrit les éventuelles erreurs ou « compiled successfully » si la compilation a réussi.

Une fois le script lancé vous pouvez l'arrêter en cliquant sur « Stop ».

Vous pouvez enregistrer votre script en faisant « Fichier/Enregistrer » et le rouvrir en faisant « Fichier/Ouvrir ».

Vous pouvez intégrer le chemin d'un fichier ou d'un dossier formaté en chaîne Pascal en cliquant respectivement sur « Fichier/Intégrer un fichier dans le code » et « Fichier/Intégrer un dossier dans le code ».

Premiers programmes

Le programme Bonjour.

Un programme est une suite d'instructions (= choses à faire par le programme).

Exemple ce programme :

```
PROGRAM bonjour ;  
  
BEGIN  
  
    showMessage('Bonjour') ;  
  
END.
```

Un programme commence par le mot clé « PROGRAM » puis le nom du programme (ici « bonjour ») puis on a un « bloc », un bloc est un bloc d'instructions qui vont ensemble, il commence par « Begin » et se termine par « End ». Ici comme c'est le bloc du programme entier, on met un point (« . ») à la fin du bloc, sinon on met un point-virgule (« ; »). Dans le bloc d'instructions du

programme se trouvent toutes les instructions du programme. Ici nous n'avons qu'une instruction « showMessage('Bonjour') ». C'est en fait une fonction qui affiche un message « Bonjour » mais nous verrons cela plus tard.

A noter qu'en Pascal on n'est pas obligé de respecter les majuscules/minuscules dans les noms de fonction ou les mots-clés.

Un nom de programme ou de fonction doit commencer par une lettre suivie de lettres, d'underscores (caractère « _ ») ou de chiffres, mais ne doit pas contenir d'accents, de caractères spéciaux ou d'espaces.

Chaque instruction doit obligatoirement se terminer par un point-virgule.

Commentaires

Dans un programme on peut mettre des « commentaires », ce sont des textes qui ne s'exécutent pas (donc ne sont pas des instructions) mais servent à mieux comprendre le programme. Si vous voulez mettre en commentaire une ligne mettez « // » devant comme ici :

```
PROGRAM bonjour ;  
  
BEGIN  
    // affiche bonjour  
    showMessage('Bonjour') ;  
  
END.
```

Si vous voulez mettre des commentaires sur plusieurs lignes mettez un « { » avant le commentaire et un « } » après comme ici :

```
PROGRAM bonjour ;  
  
BEGIN  
    { affiche  
    Bonjour}  
    showMessage('Bonjour') ;  
  
END.
```

Les variables

Une variable est une zone mémoire (vous pouvez le comprendre comme une boîte ») avec :

- Un nom : commençant par une lettre suivie de lettres, d'underscores ou de chiffres, ne doit pas contenir d'accents, de caractères spéciaux ou d'espaces
- Un type : par exemple « integer » pour un nombre entier, nous allons les voir plus en détail après
- Une valeur : la valeur contenue dans la boîte

Voici les principaux types d'une variable :

- Integer : un nombre entier
- Double : un nombre à virgule
- Char : un caractère
- String : une « chaîne de caractère » autrement dit un texte, une suite de caractères quoi
- Boolean : c'est un type spécial, il vaut soit « true » (pour « vrai ») soit false (pour « faux ») et ça peut être le résultat d'une opération par exemple $2 > 3$ renvoie « false » et $3 > 2$ renvoie « true »

Dans le programme un « char » ou un « string » doivent être entourés de guillemets simples (« ' »).

Une variable doit être « déclarée » avant d'être utilisée, pour cela on met son nom suivi de « : » suivi de son type et d'un point-virgule entre le mot-clé « var » et le mot-clé « begin » du programme.

Exemple :

```
PROGRAM exemple_variable ;
```

```
VAR
```

```
    S : string ; // déclaration d'une variable de nom « S » de type « string »
```

```
BEGIN
```

```
    S := 'Bonjour' ; //affectation de la chaîne de caractère 'Bonjour' à la variable « S »
```

```
    showMessage(S) ; // affichage du contenu de la variable « S »
```

```
END.
```

Pour donner une valeur à la variable on utilise l'opérateur « := », avec à gauche le nom de la variable et à droite la valeur. On dit que l'on « affecte » la valeur à la variable.

Identificateur

On a déjà vu que les noms des variables ou des fonctions devaient respecter des règles, en voici un rappel :

Ces noms doivent commencer par une lettre suivie de lettres, d'underscores (« _ ») ou de chiffres, ne doit pas contenir d'accents, de caractères spéciaux ou d'espaces, ils ne doivent pas non plus correspondre à des « mots-clés ». Les mots-clés sont des mots spéciaux du langage Pascal qui sont par exemple « Begin », « Program », « Var » etc.

2. Les types prédéfinis

Type entier (integer)

Les integer vont de -2147483648 à +2147483647, c'est largement suffisant pour une utilisation courante.

On peut faire des opérations sur ces entiers par exemple :

« + » : ajoute les entiers

« - » : soustrait deux entiers

« / » : divise deux entiers

« * » : multiplie deux entiers

« div » : dividende de la division entière

« mod » : reste de la division entière

Exemple :

```
PROGRAM test_entiers ;
```

```
VAR
```

```
Entier1, Entier2, Resultat : integer ;
```

```
BEGIN
```

```
    Entier1 :=1 ;
```

```
    Entier2 :=2 ;
```

```
    Resultat :=Entier1+Entier2 ;
```

```
    showMessage(intToStr(Resultat)) ;
```

```
END.
```

Ici nous déclarons deux entiers (Entier1, et Entier2), puis nous leur affectons des valeurs, puis nous calculons leur somme que nous mettons dans la variable « Resultat ». Puis nous affichons le résultat. Attention ! nous mettons ici la fonction « intToStr » avant, car il faut convertir « Resultat » en « string » avant de le passer à la fonction « showMessage » car en effet « showMessage » ne prend que des « string » en paramètre mais nous verrons cela plus en détail plus tard.

Type réel (double)

Cela fonctionne comme les integer sauf que les bornes sont beaucoup plus grandes et qu'on peut avoir des chiffres après la virgule. Les opérations sont : + , - , / et * mais pas bien sûr mod et div.

Type caractère (char)

C'est tout caractère autrement dit symbole du clavier (lettre, chiffre, caractère spécial, espace).

Dans le programme il doit être enserré de « ' ».

Type chaîne de caractères (string)

C'est une suite de caractères comme « 'Bonjour' », dans le programme doit être enserré de « ' ». Si la chaîne contient un « ' » dans le programme on doit mettre deux « ' » exemple :

```
Phrase := 'Il s'est fait mal !'
```

Il y a un opérateur spécial dit de « concaténation » qui permet d'aboutir deux chaînes de caractères, exemple :

```
showMessage('Bonjour'+ ' Le Monde !');
```

```
affichera « Bonjour Le Monde !» ;
```

L'opérateur « length » permet de récupérer le nombre de caractères d'une chaîne, exemple :

`showMessage(intToStr(length('Bonjour')))` ; affichera « 7 ».

Type booléen (boolean)

Il ne peut prendre que deux valeurs :

- Vrai (true)
- Faux (false)

Avec ce type on a trois opérateurs :

- Le ou (« or »)
- Le et (« and »)
- Le non (« not »)

Voici le résultat de ces opérateurs dans ce qu'on appelle la « table de vérité » :

| x | y | not x | x and y | x or y |
|-------|-------|-------|---------|--------|
| true | true | false | true | true |
| true | false | false | false | true |
| false | true | true | false | true |
| false | false | true | false | false |

Le résultat d'une comparaison par les opérateurs : =, >= (supérieur ou égal), <, <= (inférieur ou égal) et <> (différent) est un booléen.

Exemple :

Booleen := (21>3) and (1<=3)

« Booleen » vaut « true »

ATTENTION : vous devez obligatoirement mettre des parenthèses autour des éléments comparés

Fonctions et procédures

Les fonctions et procédures sont des petits sous-programmes qui exécutent des instructions. La différence entre les deux est qu'une fonction renvoie un résultat tandis qu'une procédure ne renvoie rien. Elles prennent des arguments ou paramètres qui sont des valeurs pour leur servir mais ce n'est pas obligatoire.

Exemple :

La procédure « showMessage » prend en paramètre un string qui est le message à afficher, c'est une procédure car il n'y a pas de résultat.

La fonction « sin » (sinus) est une fonction qui prend en paramètre un nombre réel et qui renvoie un nombre réel.

Exemple d'appel :

`showMessage('bonjour !')` ;

```
truc := sin(30) ;
```

comme vous pouvez le remarquer les paramètres sont entre parenthèse après le nom de la fonction ou de la procédure et sont séparés s'ils sont plusieurs par des virgules.

Les « branchements »

Le « if »

C'est une instruction pour dire « si telle condition est vraie alors faire telle chose »

Exemple :

```
If maVar > 3 then
```

```
    showMessage ('Bonjour !') ;
```

ici si le contenu de « maVar » est supérieur à 3 alors le message « Bonjour » sera affiché

On peut mettre un « sinon » (« else ») ainsi :

```
If maVar > 3 then
```

```
    showMessage ('Bonjour !')
```

```
else
```

```
    showMessage ('Au revoir !') ;
```

ici si le contenu de « maVar » est supérieur à 3 alors le message « Bonjour » sera affiché sinon ça affichera « Au revoir ! ».

On peut les imbriquer et même faire des « paquets » avec des « begin end »

Exemple :

```
If maVar = 2 then
```

```
Begin
```

```
    If maVar2 = 3 then
```

```
        showMessage ('maVar = 2 and maVar2 = 3 !') ;
```

```
    else
```

```
        showMessage ('maVar = 2 and maVar2 <> 3 !') ;
```

```
end
```

```
else if maVar > 2 then
```

```
    showMessage ('maVar > 2') ;
```

```
else
```

```
    showMessage ('maVar < 2') ;
```

Vous remarquerez qu'il ne faut jamais de « ; » après le « then » ou avant le « else ».

Le « case »

Le « case » permet de donner une variable et de faire différentes instructions selon ses valeurs.

Exemple :

Case maVar of

0 : showMessage('maVar=0') ; //une seule instruction, pas besoin de « begin, end »

1 : begin

maVar2=3 ;

showMessage('maVar=1') ; //plusieurs instructions, besoin « begin end »

end ;

2 : showMessage('maVar=2') ;

3, 4 : showMessage('maVar=3 ou maVar=4') ; //plusieurs valeurs : virgule

10..20 : showMessage('maVar est dans l'intervalle [10,20]') ; //intervalle : deux-points

else showMessage('maVar<>0 et maVar<>1 et maVar<>2 et maVar<>3 et maVar<>4 et maVar n'est pas dans l'intervalle [10,20]') ; //else : tous les autres cas

end ;

Cette instruction signifiant choix selon permet d'exécuter l'une des instructions selon le cas E. E est peut être entier, un caractère, un booléen mais pas un réel ni une chaîne de caractères). Les constantes du choix doivent devant des « : ». Comment ça marche E est évalué. Ensuite, E est recherchée parmi les valeurs possibles, laquelle est égale à E. L'instruction correspondante est alors exécutée. Sinon, l'instruction après le « else » (s'il y en a un) est exécutée.

On peut donner une liste de constantes, ou des intervalles de constantes. Attention, chaque valeur possible ne doit être représentée qu'une fois au plus (sinon il y a erreur à la compilation). Par exemple, on ne peut pas faire des intervalles se chevauchant, comme 3..6 et 5..10, les cas 5 et 6 étant représentés 2 fois.

Les boucles

La boucle « while »

Cette instruction permet de répéter une instruction ou un bloc d'instructions **tant que** la condition est vraie (=true).

A noter que le test de la condition est fait avant l'instruction « while » donc si le test est faux dès le début l'instruction while n'est jamais exécutée.

Exemple :

maVar :=0 ;

While maVar<3 do

maVar :=maVar +1 ;

Ici la boucle sera exécutée 3 fois.

La syntaxe est while suivi de la condition qui doit être de type booléenne.

Si la condition est toujours vraie le programme boucle à l'infini et ne s'arrête jamais, c'est ce qu'on appelle une « boucle infinie », c'est à éviter, exemple :

```
While true ;
```

La boucle repeat

Cette instruction permet de répéter une instruction ou un bloc d'instructions **jusqu'à ce que** la condition est vraie (=true).

A noter que le test de la condition est fait après l'instruction « repeat » donc si le test est faux dès le début l'instruction while sera quand même exécuté une fois.

Exemple :

```
maVar :=0 ;
```

```
repeat
```

```
    maVar :=maVar +1 ;
```

```
until maVar=3 ;
```

La syntaxe est repeat suivi d'instructions suivi de until (« jusqu'à » en anglais) puis de la condition qui doit être de type booléenne.

Contrairement au « while » pas besoin de « begin, end » le bloc « repeat until » suffit.

La boucle « for »

Permet de répéter un bloc d'instructions un certain nombre de fois.

Exemple :

```
For i :=1 to 3 do
```

```
    showMessage(inttostr(i)) ;
```

cette boucle va afficher trois messages avec les messages « 1 », puis « 2 » puis « 3 ».

Syntaxe : écrire for, puis le nom de la variable qui va changer à chaque tour de boucle (appelée « variable de boucle » (ici « i »), puis « := » puis la valeur de départ de cette variable puis « to » puis la valeur de fin de cette variable puis « do ». A chaque tour de boucle la variable de boucle va être « incrémentée » (c'est-à-dire que l'on lui ajoute 1). Quand la variable de boucle est égale à la valeur de fin la boucle se termine.

A noter que l'on peut mettre un bloc « begin, end » après l'instruction « for ».

A noter qu'il y a l'équivalent pour décrémenter la variable de boucle :

```
For i :=3 downto 1 do
```

```
    showMessage(inttostr(i)) ;
```

cette boucle va afficher trois messages avec les messages « 3 », puis « 2 » puis « 1 ».

Le break et le continue

L'instruction « break » dans une boucle termine aussitôt la boucle, exemple :

```
While true do
```

```
Begin
```

```
    If maVar>3 then
```

```
        Break ;//termine la boucle si maVar > 3
```

```
    maVar :=maVar+1 ;
```

```
end ;
```

L'instruction « continue » saute le tour de boucle actuel et passe automatiquement au suivant, exemple :

```
For i :=0 to 5 do
```

```
Begin
```

```
    If i=0 then
```

```
        Continue ;//saute le tour de boucle 0
```

```
    showMessage(intToStr(i)) ;
```

```
End ;
```

Le type tableau

Un tableau est une série de variables dans un certain ordre.

Exemple :

```
Var tableau :array[1..3] of integer ;
```

Crée un tableau de 3 valeurs.

Après on peut initialiser ces valeurs comme ceci :

```
Tableau[1] :=1 ;
```

```
Tableau[2] :=3 ;
```

```
Tableau[3] :=5 ;
```

Puis l'on peut afficher le contenu de ces cases avec par exemple :

```
ShowMessage(intToStr(tableau[1])) ; affichera « 3 »
```

On peut faire des tableaux multidimensionnels par exemple :

```
Var tableau :array[1..3,1..4] of integer ;
```

Sera un tableau de 3 lignes et 4 colonnes, après on accède aux cases comme ceci : tableau[ligne, colonne] par exemple tableau[2,3]

On peut même faire des tableaux de 3 dimensions ou plus !

Les exceptions

Les exceptions sont des erreurs qui se passent lors de l'exécution du programme, pour ne pas planter le programme lors de son exécution on peut enserrer la partie où se produit l'exception et gérer l'exception dans le programme, exemple :

Try

```
maVar := 1/0 ;//exception : division par 0
```

except

```
showMessage('Erreur !');
```

end ;

La syntaxe est la suivante : entre les mots-clés « try » et « except » mettez le code à protéger et entre les mots-clés « except » et « end » mettez le code à faire dans le cas d'une erreur. Si une erreur se produit dans le bloc « try except » l'exécution saute directement dans le bloc « except end » sinon le bloc « except end » n'est pas exécuté.

Les imbrications

On peut imbriquer des instructions par exemple :

While maVar >3 do

```
If maVar<2 then
```

```
showMessage(inttostr(maVar));
```

Les fonctions communes à DrawScript et RadioScript

D'abord ce que vous devez savoir c'est qu'il y a deux variables globales cachées :

- « fichiers » qui contient un ensemble de fichiers
- « pwd » qui contient le répertoire courant

Liste des fonctions :

```
function random(n :integer) :integer ;
```

Renvoie un nombre aléatoire entre 0 et n-1

```
function intToStr(i :integer) :string ;
```

Convertit un integer en string

```
function strToInt(i :integer) :integer ;
```

Convertit un string en integer

```
procedure showMessage(s: string);
```

Affiche une boîte de message avec le message « s ».

Function `inputBox(windowTitle :string ;prompt :string ;default :string) :string ;`

Affiche une boîte de dialogue de saisie de chaîne avec comme titre de fenêtre « `windowTitle` », comme texte de demande de saisie par défaut : « `prompt` » et comme texte par défaut dans la boîte de saisie : « `default` ».

function `getFolderPath:string;`

Ouvre une fenêtre de dialogue de choix d'un dossier et retourne son chemin

procedure `setPwd(s: string);`

Mets le répertoire courant (`pwd`) à `s`

function `getPwd:string;`

Retourne `pwd`

procedure `listFiles(recursive:boolean;includeDirs:boolean;`

`includeFiles:boolean;path:string;fileMask:string ;addFiles :boolean) ;`

Liste les chemins complets des fichiers et/ou des dossiers du répertoire « `path` » et les met dans la variable « `fichiers` ». Si `addFiles` vaut `true` le contenu de la variable « `fichiers` » est écrasé sinon les fichiers sont ajoutés à la fin de cette variable. Il comprend les sous-dossiers récursivement si « `recursive` » vaut `true`. Si « `includeDirs` » vaut `true` les chemins dossiers sont retournés. Si « `includeFiles` » est à `true`, les chemins des fichiers sont retournés. « `fileMask` » est un masque du nom de fichier utilisant les caractères suivants :

- `*` : un ou plusieurs caractères
- `|` : « ou »

procedure `shuffle;`

Mélange les fichiers.

procedure `showFiles;`

Montre dans une fenêtre modale la liste des fichiers de la variable « `fichiers` ».

function `getFilesCount:integer;`

Retourne le nombre de chemins de fichiers dans la variable « `fichiers` ».

function `getFile(i:integer):string;`

Retourne le fichier numéro « `i` » dans la liste des fichiers « `fichiers` ».

Fonction spécifique à RadioScript

procedure `play(son:string);`

Joue le son de chemin de fichier « `son` ».

Fonctions mathématiques



Fonctions mathématiques de base

- **pi** : Cette constante représente le nombre π (environ 3,14159), essentiel pour les calculs circulaires et trigonométriques.
- **min(a, b)** : Retourne la plus petite valeur entre a et b.
- **max(a, b)** : Retourne la plus grande valeur entre a et b.
- **power(base, exposant)** : Élève base à la puissance exposant. Par exemple, `power(2, 3)` retourne 8.
- **sqr(x)** : Retourne le carré de x, équivalent à `power(x, 2)`.
- **sqrt(x)** : Retourne la racine carrée de x. Par exemple, `sqrt(9)` retourne 3.



Fonctions trigonométriques

Ces fonctions sont essentielles pour travailler avec des angles et des formes géométriques. Elles utilisent des angles exprimés en **radians**. Pour convertir des degrés en radians, utilise la formule : $\text{radians} = \text{degrees} * (\pi / 180)$.

- **cos(angle)** : Retourne le cosinus de l'angle donné en radians.
- **sin(angle)** : Retourne le sinus de l'angle donné en radians.
- **tan(angle)** : Retourne la tangente de l'angle donné en radians.
- **arccos(x)** : Retourne l'angle en radians dont le cosinus est x.
- **arcsin(x)** : Retourne l'angle en radians dont le sinus est x.
- **arctan(x)** : Retourne l'angle en radians dont la tangente est x.
- **arctan2(y, x)** : Retourne l'angle en radians entre l'axe des X et le point (x, y), utile pour déterminer l'angle d'un vecteur.

Fonctions d'arrondi

Ces fonctions permettent d'ajuster les nombres à l'entier le plus proche ou selon des règles spécifiques.

- **floor(x)** : Arrondit x à l'entier inférieur le plus proche. Par exemple, floor(3.7) retourne 3.
- **ceil(x)** : Arrondit x à l'entier supérieur le plus proche. Par exemple, ceil(3.2) retourne 4.
- **round(x)** : Arrondit x à l'entier le plus proche. Si la partie décimale est de 0,5 ou plus, l'arrondi se fait vers le haut ; sinon, vers le bas. Par exemple, round(3.5) retourne 4, et round(3.4) retourne 3.

Fonctions de chaîne (1ère partie)

Les fonctions de chaîne permettent de **manipuler les textes** : extraire des morceaux, les transformer, chercher des mots, etc.

Voici les premières fonctions que tu peux utiliser dans tes programmes RadioScript ou DrawScript.

Qu'est-ce que la "casse" ?

En informatique, on appelle "**casse**" la **différence entre les lettres majuscules et les lettres minuscules**.

Par exemple, les textes suivants n'ont **pas la même casse** :

- bonjour
- Bonjour
- BONJOUR

Même s'ils contiennent les **mêmes lettres**, pour l'ordinateur, ces trois mots **ne sont pas identiques** si on tient compte de la casse.

Quand c'est important ?

- Si une fonction est **sensible à la casse**, alors chat et CHAT sont considérés comme **différents**.
 - Si elle est **insensible à la casse**, alors chat, CHAT, ChAt... sont tous vus comme **identiques**.
-

✅ Un petit tableau pour bien comprendre :

Texte 1 Texte 2 Égaux si casse respectée ? Égaux si casse ignorée ?

| | | | |
|-------|-------|-------|-------|
| Chat | chat | ❌ Non | ✅ Oui |
| chien | chien | ✅ Oui | ✅ Oui |
| HELLO | hello | ❌ Non | ✅ Oui |

Conseil du robot

Quand tu compares des textes, pense toujours :

Est-ce que la casse est importante ici ?

Par exemple :

- Pour comparer deux mots **insensiblement à la casse**, utilise `AnsiCompareText`, `AnsiContainsText`, `AnsiStartsText`...
- Pour une comparaison stricte, utilise `CompareStr`, ou même `Pos` (qui est sensible à la casse).

C'est quoi une chaîne de caractères ?

Une **chaîne de caractères** (ou *string* en anglais) est un **texte**, c'est-à-dire une suite de **lettres**, de **chiffres**, de **symboles**, ou même d'espaces.

Par exemple, ce sont toutes des chaînes de caractères :

- 'Bonjour'
- '12345'
- 'Salut les amis !'
- 'a+b=c'

Comment ça fonctionne ?

En informatique, une chaîne de caractères est comme un **collier de perles**, où chaque **lettre est une perle** placée à une position précise.

Par exemple, dans le mot 'Salut' :

Position Lettre

| | |
|---|---|
| 1 | S |
| 2 | a |
| 3 | l |
| 4 | u |
| 5 | t |

Tu peux :

- **compter** combien de lettres il y a avec Length
- **extraire** une partie avec Copy ou MidStr
- **chercher** un mot dedans avec Pos ou AnsiContainsText
- **changer** les majuscules/minuscules
- **remplacer** du texte

Conseil du robot

Les chaînes sont **entre guillemets simples** en Pascal, comme ceci : 'Coucou'.
N'oublie pas les ' sinon le programme ne comprendra pas que c'est du texte !

1. Copy

Ce que fait la fonction :

Copie une partie d'une chaîne de caractères à partir d'une position donnée.

Paramètres :

- S : le texte de départ
- Index : la position de départ (le premier caractère est à 1)
- Count : le nombre de caractères à copier

Exemple :

```
program ExempleCopy;  
  
var  
  
    texte, extrait: string;  
  
begin  
  
    texte := 'Bonjour tout le monde';  
  
    extrait := Copy(texte, 9, 4); // Extrait 'tout'  
  
    ShowMessage(extrait);  
  
end.
```

2. MidStr

Ce que fait la fonction :

Comme Copy, elle extrait un morceau de texte à partir d'une position.

Paramètres :

- S : la chaîne de départ
- Start : la position de départ (1 = premier caractère)
- Count : le nombre de caractères à prendre

Exemple :

```
program ExempleMidStr;  
  
var  
  
    phrase: string;  
  
begin  
  
    phrase := MidStr('Programmation', 5, 6); // => 'rammat'  
  
    ShowMessage(phrase);  
  
end.
```

◆ 3. Pos

✅ Ce que fait la fonction :

Cherche une sous-chaîne dans une autre et renvoie sa position (ou 0 si elle n'est pas trouvée).

✖ Paramètres :

- SubStr : le texte à chercher
- S : le texte dans lequel on cherche

🔍 Exemple :

```
program ExemplePos;  
  
var  
  
    position: Integer;  
  
begin  
  
    position := Pos('chat', 'Le chat dort');  
  
    ShowMessage('Position : ' + IntToStr(position)); // Affiche 4  
  
end.
```

◆ 4. AnsiReplaceStr

✅ Ce que fait la fonction :

Remplace un texte par un autre dans une chaîne (sensible à la casse).

✖ Paramètres :

- S : le texte d'origine
- OldPattern : ce qu'on veut remplacer
- NewPattern : ce qu'on met à la place

Exemple :

```
program ExempleReplace;  
  
var  
  
    phrase: string;  
  
begin  
  
    phrase := AnsiReplaceStr('Vive le chocolat', 'chocolat', 'fromage');  
  
    ShowMessage(phrase); // Affiche : Vive le fromage  
  
end.
```

5. AnsiLowerCase

Ce que fait la fonction :

Transforme tout le texte en **minuscules**, y compris les accents.

Paramètre :

- S : le texte à transformer

Exemple :

```
program ExempleMin;  
  
begin  
  
    ShowMessage(AnsiLowerCase('BONJOUR À TOUS'));  
  
end.
```

6. AnsiUpperCase

Ce que fait la fonction :

Transforme tout le texte en **majuscules**, y compris les accents.

Paramètre :

- S : le texte à transformer

Exemple :

```
program ExempleMaj;  
  
begin  
  
  ShowMessage(AnsiUpperCase('vive le chocolat !'));  
  
end.
```

7. Trim

Ce que fait la fonction :

Supprime les **espaces au début et à la fin** d'un texte.

Paramètre :

- S : le texte à nettoyer

Exemple :

```
program ExempleTrim;  
  
begin  
  
  ShowMessage(Trim(' coucou ')); // Affiche : 'coucou'  
  
end.
```

8. TrimLeft

Ce que fait la fonction :

Supprime les espaces à **gauche** du texte (au début).

Paramètre :

- S : le texte à nettoyer

Exemple :

```
program ExempleTrimLeft;  
  
begin  
  
  ShowMessage(TrimLeft(' salut')); // Affiche : 'salut'  
  
end.
```

◆ 9. TrimRight

✓ Ce que fait la fonction :

Supprime les espaces à **droite** du texte (à la fin).

✖ Paramètre :

- S : le texte à nettoyer

🔍 Exemple :

```
program ExempleTrimRight;  
  
begin  
  
  ShowMessage(TrimRight('salut ')); // Affiche : 'salut'  
  
end.
```

◆ 10. LeftStr

✓ Ce que fait la fonction :

Prend les **X premiers caractères** d'un texte.

✖ Paramètres :

- S : le texte d'origine
- Count : le nombre de caractères à garder au début

🔍 Exemple :

```
program ExempleLeftStr;  
  
begin  
  
  ShowMessage(LeftStr('Abracadabra', 4)); // Affiche : 'Abra'  
  
end.
```

🤖 Conseil du robot

Tu peux combiner plusieurs fonctions ! Par exemple :

```
ShowMessage(AnsiUpperCase(Trim(Copy(S, 1, 5))));
```

Chapitre X – Fonctions de chaîne (2^e partie)

11. RightStr

Ce que fait la fonction :

Prend les **X derniers caractères** d'une chaîne.

Paramètres :

- S : le texte d'origine
- Count : le nombre de caractères à garder à la fin

Exemple :

```
program ExempleRightStr;  
  
begin  
  
    ShowMessage(RightStr('Abracadabra', 3)); // Affiche : 'bra'  
  
end.
```

12. AnsiContainsText

Ce que fait la fonction :

Vérifie si une chaîne **contient** un texte, **sans tenir compte des majuscules/minuscules**.

Paramètres :

- AText : le texte dans lequel on cherche
- ASubText : le texte à chercher

Exemple :

```
program ExempleContains;  
  
begin  
  
    if AnsiContainsText('Programmation en Pascal', 'pascal') then  
  
        ShowMessage('Oui, Pascal est bien là !');  
  
end.
```

◆ 13. ReverseString

✓ Ce que fait la fonction :

Renverse l'ordre des lettres dans le texte.

✖ Paramètre :

- S : le texte à inverser

🔍 Exemple :

```
program ExempleReverse;
```

```
begin
```

```
  ShowMessage(ReverseString('Bonjour')); // Affiche : 'ruojnoB'
```

```
end.
```

◆ 14. DupeString

✓ Ce que fait la fonction :

Répète un même texte plusieurs fois.

✖ Paramètres :

- S : le texte à dupliquer
- Count : combien de fois le répéter

🔍 Exemple :

```
program ExempleDupe;
```

```
begin
```

```
  ShowMessage(DupeString('Ha', 3)); // Affiche : 'HaHaHa'
```

```
end.
```

◆ 15. StartsText

✓ Ce que fait la fonction :

Vérifie si une chaîne **commence** par un texte donné (sans tenir compte de la casse).

Paramètres :

- ASubText : le début attendu
- AText : le texte complet

Exemple :

```
program ExempleStartsText;  
  
begin  
  
  if StartsText('salut', 'Salut les amis') then  
  
    ShowMessage('La phrase commence bien par "salut"');  
  
end.
```

16. EndsText

Ce que fait la fonction :

Vérifie si une chaîne **se termine** par un texte donné (sans tenir compte de la casse).

Paramètres :

- ASubText : le texte attendu à la fin
- AText : le texte complet

Exemple :

```
program ExempleEndsText;  
  
begin  
  
  if EndsText('monde', 'Bonjour le monde') then  
  
    ShowMessage('La phrase se termine bien par "monde"');  
  
end.
```

17. AnsiStartsText / AnsiEndsText

Ce que font ces fonctions :

Même rôle que StartsText et EndsText, mais utilisent une méthode différente (compatibilité accentuées, plus fiable avec certains caractères spéciaux).

Paramètres :

- ASubText : début ou fin à comparer
- AText : texte à tester

Exemple commun :

```
program ExempleAnsiStartEnd;  
  
begin  
  
  if AnsiStartsText('Bonjour', 'Bonjour tout le monde') then  
  
    ShowMessage('Ça commence bien !');  
  
  if AnsiEndsText('monde', 'Bonjour tout le monde') then  
  
    ShowMessage('Ça finit bien aussi !');  
  
end.
```

18. AnsiStartsStr / AnsiEndsStr

Ce que font ces fonctions :

Comme ci-dessus, mais la comparaison est **sensible à la casse** !

Paramètres :

- ASubText : le texte à chercher au début ou à la fin
- AText : le texte principal

Exemple :

```
program ExempleStrict;  
  
begin  
  
  if AnsiStartsStr('Bonjour', 'Bonjour tout le monde') then  
  
    ShowMessage('C'est bien un "Bonjour" majuscule.');
```

end.

◆ 19. AnsiCompareText

✓ Ce que fait la fonction :

Compare deux textes **sans tenir compte de la casse**.

Renvoie 0 s'ils sont identiques, un nombre négatif ou positif sinon.

✖ Paramètres :

- S1 : premier texte
- S2 : second texte

🔍 Exemple :

```
program ExempleCompareText;  
  
begin  
  
  if AnsiCompareText('pomme', 'POMME') = 0 then  
  
    ShowMessage('Les deux mots sont les mêmes (sans casse)');  
  
end.
```

◆ 20. CompareStr

✓ Ce que fait la fonction :

Compare deux textes **avec sensibilité à la casse**.

✖ Paramètres :

- S1 : premier texte
- S2 : second texte

🔍 Exemple :

```
program ExempleCompareStr;  
  
begin  
  
  if CompareStr('Test', 'test') <> 0 then  
  
    ShowMessage('Ils sont différents à cause de la casse.');
```

end.

◆ 21. IfThen

✅ Ce que fait la fonction :

Renvoie un texte **en fonction d'une condition** : si c'est vrai, on renvoie un, sinon l'autre.

✖ Paramètres :

- ACondition : un test (vrai ou faux)
- ATrue : ce qu'on renvoie si c'est vrai
- AFalse : ce qu'on renvoie si c'est faux

🔍 Exemple :

```
program ExempleIfThen;

var

  estGrand: Boolean;

  message: string;

begin

  estGrand := True;

  message := IfThen(estGrand, 'Tu es grand !', 'Tu es petit...');

  ShowMessage(message);

end.
```

🤖 Conseil du robot

Tu peux faire des combinaisons rigolotes :

```
ShowMessage(ReverseString(UpperCase('Bonjour')));
```

Tu veux du fun ? Essaie de faire une phrase **à l'envers** ou une **répétition infinie** avec DupeString (mais attention à ne pas bloquer le programme ! 😊)

◆ 22. LowerCase

✅ Ce que fait la fonction :

Transforme tout le texte en **minuscules**. Contrairement à AnsiLowerCase, celle-ci **ne gère pas les accents**, seulement les lettres A-Z.

Paramètre :

- S : le texte à transformer

Exemple :

```
program ExempleLowerCase;  
  
begin  
  
  ShowMessage(LowerCase('HELLO WORLD!')); // Affiche : hello world!  
  
end.
```

23. UpperCase

Ce que fait la fonction :

Transforme tout le texte en **majuscules**. Comme LowerCase, elle ne gère pas les caractères accentués.

Paramètre :

- S : le texte à transformer

Exemple :

```
program ExempleUpperCase;  
  
begin  
  
  ShowMessage(UpperCase('bonjour à tous')); // Affiche : BONJOUR À TOUS  
  
end.
```

24. Length

Ce que fait la fonction :

Donne le **nombre de caractères** dans une chaîne (y compris les espaces, les lettres accentuées, etc.).

Paramètre :

- S : la chaîne dont on veut connaître la taille

Exemple :

```
program ExempleLength;  
  
var  
  
    nb: Integer;  
  
begin  
  
    nb := Length('Coucou !');  
  
    ShowMessage('Il y a ' + IntToStr(nb) + ' caractères.');
```

end.

Conseil du robot

Tu peux utiliser Length pour faire des boucles ou vérifier si un texte est vide :
if Length(S) = 0 then ShowMessage('Le texte est vide !');

Afficher des messages et jouer des sons

Dans ce chapitre, tu vas apprendre à utiliser deux fonctions **super utiles** pour **parler avec l'utilisateur** :

- MessageBox : pour afficher des **fenêtres avec des boutons**
 - MessageBeep : pour jouer un **petit son** Windows (comme une alerte ou un bip joyeux)
-

25. MessageBox

Ce que fait la fonction :

Affiche une **fenêtre avec un message**, un titre, des **boutons** et des **icônes**.

Paramètres :

- Text : le message affiché
- Caption : le titre de la fenêtre
- Flags : des options pour choisir les boutons et l'icône

Exemples de constantes pour Flags :

| Constante | Signification |
|--------------------|--|
| MB_OK | Un seul bouton "OK" |
| MB_YESNO | Deux boutons "Oui" et "Non" |
| MB_OKCANCEL | Boutons "OK" et "Annuler" |
| MB_ICONINFORMATION | Icône d'information (i) |
| MB_ICONWARNING | Icône attention (triangle jaune) |
| MB_ICONERROR | Icône rouge erreur (croix) |
| MB_ICONQUESTION | Icône question (?) |
| MB_DEFBUTTON2 | Deuxième bouton sélectionné par défaut |

Constantes de retour :

Constante Signification

| | |
|----------|-----------------------|
| IDOK | Bouton OK cliqué |
| IDYES | Bouton Oui cliqué |
| IDNO | Bouton Non cliqué |
| IDCANCEL | Bouton Annuler cliqué |

Exemple simple :

```
program ExempleMessageBox;
var
    choix: Integer;
begin
    choix := MessageBox('Veux-tu continuer ?', 'Question', MB_YESNO or MB_ICONQUESTION);
    if choix = IDYES then
        ShowMessage('Tu as choisi OUI !')
    else
        ShowMessage('Tu as choisi NON...');
```

end.

Et le mot or alors ?

Quand on écrit quelque chose comme :



MB_YESNO or MB_ICONQUESTION

Cela veut dire qu'on **combine plusieurs options** en une seule.

Ici, on veut à la fois :

☒ Des boutons "**Oui**" et "**Non**"

☒ Une icône "?" (**question**)

 En informatique, le mot or ne veut pas dire "ou bien l'un ou l'autre", mais "**ajouter une option en plus**". C'est un peu comme empiler des cases à cocher 

26. MessageBeep

☒ Ce que fait la fonction :

Joue un **petit son Windows** (bip d'alerte, succès, erreur...).

Paramètre :

- BeepType : le type de son à jouer (utilise une constante)

Constantes pour le son :

| Constante | Son joué |
|--------------------|--------------------------------|
| MB_ICONASTERISK | Bip doux (info) |
| MB_ICONEXCLAMATION | Bip d'alerte jaune |
| MB_ICONHAND | Bip d'erreur (rouge) |
| MB_ICONQUESTION | Bip "?" |
| 0 | Son par défaut (bip classique) |

Exemple simple :

```
program ExempleBeep;
```

```
begin
```

```
    MessageBeep(MB_ICONEXCLAMATION); // Joue un bip d'alerte
```

```
end.
```

Conseil du robot

Tu peux combiner MessageBox et MessageBeep pour faire une boîte qui **parle et sonne** :

```
MessageBeep(MB_ICONQUESTION);
```

```
MessageBox('Veux-tu vraiment quitter ?', 'Attention', MB_YESNO or MB_ICONQUESTION);
```

Programme complet :

```
program FonctionsDeChaine;
```

```
var
```

```
    texte, extrait, resultStr: string;
```

```
    posi, longueur: Integer;
```

```
    estPresent: Boolean;
```

```
    choix: Integer;
```

```
begin
```

```
    // 1. Copy
```

```
    texte := 'Bonjour les amis !';
```

```
    extrait := Copy(texte, 9, 3); // => 'les'
```

```
    ShowMessage('Copy : ' + extrait);
```

```
    // 2. MidStr
```

```
    extrait := MidStr(texte, 1, 7); // => 'Bonjour'
```

```
    ShowMessage('MidStr : ' + extrait);
```

```
    // 3. Pos
```

```
    posi := Pos('amis', texte); // => 13
```

```
    ShowMessage('Pos : ' + IntToStr(posi));
```

```
// 4. AnsiReplaceStr

resultStr := AnsiReplaceStr(texte, 'amis', 'robots');

ShowMessage('AnsiReplaceStr : ' + resultStr);


// 5. AnsiLowerCase

ShowMessage('AnsiLowerCase : ' + AnsiLowerCase('BONJOUR À TOUS'));


// 6. AnsiUpperCase

ShowMessage('AnsiUpperCase : ' + AnsiUpperCase('vive le chocolat'));


// 7. Trim

ShowMessage('Trim : [' + Trim(' coucou ') + ']');


// 8. TrimLeft

ShowMessage('TrimLeft : [' + TrimLeft(' Hello') + ']');


// 9. TrimRight

ShowMessage('TrimRight : [' + TrimRight('Salut ') + ']');


// 10. LeftStr

ShowMessage('LeftStr : ' + LeftStr('Abracadabra', 4));


// 11. RightStr

ShowMessage('RightStr : ' + RightStr('Abracadabra', 3));


// 12. AnsiContainsText

estPresent := AnsiContainsText('La Programmation en Pascal', 'pascal');

ShowMessage('AnsiContainsText : ' + IfThen(estPresent, 'Oui', 'Non'));
```



```
// 13. ReverseString
```

```
ShowMessage('ReverseString : ' + ReverseString('Bonjour'));
```

```
// 14. DupeString
```

```
ShowMessage('DupeString : ' + DupeString('Ha', 3)); // => 'HaHaHa'
```

```
// 15. StartsText
```

```
if StartsText('Bonjour', texte) then
```

```
    ShowMessage('StartsText : commence par "Bonjour"');
```

```
// 16. EndsText
```

```
if EndsText('amis !', texte) then
```

```
    ShowMessage('EndsText : se termine par "amis !"');
```

```
// 17. AnsiStartsText
```

```
if AnsiStartsText('bonjour', texte) then
```

```
    ShowMessage('AnsiStartsText : OK');
```

```
// 18. AnsiEndsText
```

```
if AnsiEndsText('AMIS !', texte) then
```

```
    ShowMessage('AnsiEndsText : OK');
```

```
// 19. AnsiStartsStr (sensible à la casse)
```

```
if AnsiStartsStr('Bonjour', texte) then
```

```
    ShowMessage('AnsiStartsStr : OK');
```

```
// 20. AnsiEndsStr (sensible à la casse)
```

```
if AnsiEndsStr('amis !', texte) then
```

```
    ShowMessage('AnsiEndsStr : OK');
```

```
// 21. AnsiCompareText
```

```
if AnsiCompareText('pomme', 'POMME') = 0 then
```

```
    ShowMessage('AnsiCompareText : identiques (sans casse)');
```

```
// 22. CompareStr
```

```
if CompareStr('Test', 'test') <> 0 then
```

```
    ShowMessage('CompareStr : différents (avec casse)');
```

```
// 23. LowerCase
```

```
ShowMessage('LowerCase : ' + LowerCase('HELLO'));
```

```
// 24. UpperCase
```

```
ShowMessage('UpperCase : ' + UpperCase('bonjour'));
```

```
// 25. Length
```

```
longueur := Length(texte);
```

```
ShowMessage('Length : ' + IntToStr(longueur));
```

```
// 26. IfThen
```

```
ShowMessage('IfThen : ' + IfThen(longueur > 10, 'Texte long', 'Texte court'));
```

```
// 27. MessageBox (avec OR)
```

```
choix := MessageBox('Veux-tu continuer ?', 'Question', MB_YESNO or MB_ICONQUESTION);
```

```
if choix = IDYES then
```

```
    ShowMessage('Tu as choisi OUI !')
```

```
else
```

```
    ShowMessage('Tu as choisi NON.');
```

```
// 28. MessageBeep
```

```
MessageBeep(MB_ICONEXCLAMATION);
```

```
end.
```

EXERCICE

Coder le jeu des 1000 francs, solution dans jeu_mille_francs.txt