

# Programmation en Vala

## Recueil d'exemples

Auteur principal : Alexandre Nouvel – [alex.nouvel@free.fr](mailto:alex.nouvel@free.fr)

Version 0.0.6

Première version (0.0.1) : 04/07/2018

Dernière mise-à-jours : 04/02/2022

# Table des matières

1. Introduction.....	5
Historique.....	5
Architecture de Vala.....	5
Approche adoptée dans ce tutoriel.....	5
2. Installation de Vala.....	5
1. Installation sous MacOS-X.....	5
2. Installation sous Linux.....	6
3. Installation sous Windows.....	6
3. Premier pas.....	6
1. Introduction.....	6
1. Utilitaires pour programmer.....	6
2. Ouverture d'un terminal.....	6
1 Sous MacOS-X.....	7
2 Sous Windows.....	7
3 Sous Linux.....	7
3. Les commandes de base du shell.....	7
4 Connaître le répertoire courant.....	7
5 Création d'un répertoire de travail et affichage du contenu.....	7
6 Se déplacer dans les répertoires.....	8
7 Création de notre fichier source et lancement de l'éditeur de texte.....	8
8 Premier exemple de programme.....	9
9 Compiler notre programme.....	10
10 Lancer l'exécutable.....	10
11 Combiner plusieurs lignes de commande en une seule.....	10
2. Quelques explications sur notre premier programme.....	10
3. Afficher du texte avec la fonction print.....	11
1. Séquences d'échappement les plus courant.....	11
2. Spécificateur de format.....	11
4. Lire une valeur entrée au clavier avec read_line - premier aperçu d'une variable.....	12
4. Les types simples.....	12
1. Le type int.....	12
1. Afficher un entier.....	12
2. Conversion d'un type int vers un type string.....	13
2. Le type booléen (bool).....	13
1. Afficher un type booléen.....	13
2. Conversion d'un type booléen vers un type string.....	13
3. Conversion d'un type string vers un type booléen.....	13
3. Le type char.....	13
2. Exemple - obtenir le caractère le plus grand.....	14
4. Le type double.....	14
5. Le type string.....	14
1. Concaténer des chaînes de caractères.....	14
2. Conversion d'un type string vers un type int.....	14
3. Variable string déclaré sur plusieurs lignes.....	14
5. Les structures de contrôle.....	15
1. Les conditions.....	15
1. Instruction if.....	15
2. Instruction switch/case.....	15
2. Les boucles.....	15
1. Boucle for.....	15
2. Boucle foreach.....	15
4. Boucle while.....	16
5. Boucle do ... while.....	16
6. Les fonctions.....	16
1. Introduction.....	16
7. Programmation objet.....	17
1. Les mutateurs (set).....	17

2. Les accesseurs (get).....	17
8. Les types évolués.....	17
1. Le type array.....	17
1. Tableau à une dimension.....	17
1. Déclaration d'un tableau à une dimension.....	17
2. Taille d'un tableau.....	17
3. Parcours d'un tableau.....	17
2. Copy d'un tableau.....	18
3. Tableau multi-dimensionnel.....	18
2. Le type enum.....	19
3. Le type DateTime.....	19
1. Obtenir la date courante.....	19
2. Obtenir le mois courant.....	20
4. Le type DateMonth.....	21
1. Comparaison de date.....	21
5. Le type List.....	22
1. Ajout d'un élément et parcours d'une liste.....	22
2. Type de retour et passage en argument.....	22
3. Retour d'un type List au sein d'une méthode de classe.....	23
6. Le type HashTable.....	23
7. Le type struct.....	23
1. Création et utilisation d'une structure.....	23
2. Copie de structure.....	24
3. Tableau de structure.....	24
1. Tableau statique avec initialisation des champs lors de la création.....	24
2. Tableau statique sans initialisation de départ.....	25
3. Fonction retournant un tableau de structure.....	25
4. Tableau de structure comme attribut d'un objet.....	26
9. Manipulation des répertoires et des fichiers.....	27
Le type File.....	27
10. Séparer le code en plusieurs fichiers – Créer ses propres bibliothèques - Namespaces.....	27
1. Introduction.....	27
11. Étude de quelques bibliothèques.....	27
1. Manipulation des fichiers avec gio.....	27
1. Création et écriture dans un fichier.....	27
2. Vérifier l'existence d'un fichier.....	28
3. Suppression d'un fichier.....	28
2. Interface graphique avec Gtk.....	28
1. Installation.....	28
1. Sous MacOS-X.....	28
2. Sous Linux.....	28
3. Sous Windows.....	28
2. Une simple fenêtre.....	28
3. Une fenêtre avec un label.....	29
4. Une fenêtre avec un label et un Bouton.....	29
5. Gestion des évènements.....	30
1. Événement du clavier.....	30
6. Label, Button et Entry.....	30
7. Les menus.....	31
2. Position des menus dans la barre principale.....	31
8. Le widget liste ou tableau – TreeView.....	33
3. Une liste simple comportant un seul élément.....	33
4. Afficher les éléments d'un TreeView.....	33
5. Accéder aux éléments d'un TreeView lors d'un clique de souris.....	34
6. Éditer une cellule d'un TreeView.....	36
7. Éditer une cellule d'un treview comportant plusieurs colonnes.....	37
9. Le widget ListBox.....	38
1. Insérer des éléments.....	38
2. Effacement et ajout d'éléments dans une liste.....	39

10. Le widget ComboBoxText.....	40
1 Obtenir le texte sélectionné.....	40
11. Le widget Notebook.....	41
12. Le widget Calendar.....	42
3. Graphismes et manipulation des différents formats avec Cairo.....	43
1. Installation de Cairo.....	43
2. Générations d'un document pdf.....	43
4. Base de donnée locale avec Sqlite.....	43
1. Introduction.....	43
2. Installation de Sqlite.....	43
3. Remarques sur Sqlite.....	43
4. Création d'une base de donnée.....	44
5. Création d'une table.....	44
6. Afficher des données.....	44
5. Rendu html avec WebKit.....	44
12. Faire appel à des fonctions du langage C.....	45
1. Code C.....	45
2. Code Vala.....	46
3. Compilation.....	46
4. Exécution.....	46
5. Résultat.....	46
13. Empaquetage de logiciel crée avec Vala.....	46
1. Empaquetage sous Windows.....	46
14. Annexe.....	48
1. Convention de nommage en Vala.....	48
2. Commandes shell élémentaires.....	48
1. Savoir dans quel répertoire on se situe - pwd.....	48
2. Afficher le contenu d'un répertoire – ls.....	48
3. Effacer l'écran – clear et reset.....	48
4. Ce déplacer dans un répertoire – cd.....	48
1 Chemin absolu.....	48
2 Chemin relatif.....	48
3 Monter d'un répertoire.....	48
5. Créer un répertoire – mkdir.....	48
6. Supprimer un répertoire – rmdir.....	49
3. Commandes Sqlite élémentaires.....	49
1. Démarrer Sqlite.....	49
2. Visualiser les bases de données.....	49
3. Afficher les tables.....	49
4. Visualiser l'architecture d'une table.....	49
5. Quitter Sqlite.....	49

# 1. Introduction

## Historique

En 1997, Miguel de Icaza et Federico Mena créent le projet GNOME (GNU Network Object Model Environment). L'objectif étant de créer un bureau entièrement libre sous les systèmes d'exploitations Linux et BSD. Ce bureau repose sur une bibliothèque graphique libre GTK+ qui avait été programmé à l'origine pour le logiciel Gimp, célèbre logiciel de retouche photo. Cette bibliothèque, bien que programmé en C, suit une logique de programmation objet.

La version 2 de Gtk+ a séparé la partie graphique de GTK et a donné naissance à une nouvelle bibliothèque appelé GObject, plus généraliste.

Celle-ci propose une classe de base qui permet de générer des signaux, des méthodes virtuelles, l'encapsulation, une gestion simplifiée de la mémoire et facilite la programmation multiplate-forme. Mais l'utilisation de GObject est particulière. Le fait que le langage C ne soit pas orienté objet peut dérouter les programmeurs habitués à utiliser des langages de plus haut niveau (C++, Java, C#).

En 2006, Jurg Billeter crée Vala qui s'influence fortement des langages Java et C#. Vala permet aux programmeurs GNOME de développer des applications plus rapidement. Il traduit un code source objet proche du Java (ou C#) directement en code C. Ce code est ensuite compilé avec gcc. Cette technique permet l'utilisation de toutes les bibliothèques C créées depuis plusieurs années. La stabilité et la rapidité d'exécution de ces bibliothèques ayant déjà fait leurs preuves, on bénéficie ainsi de l'apport de plusieurs années de travail fourni par des milliers de bénévoles à travers le monde.

Vala est multiplate-forme, il s'exécute sans problème sous MacOS-X, Linux et Windows.

## Architecture de Vala

Vala s'appuie essentiellement sur deux bibliothèques provenant du C : la Glib et GObject.

- La Glib permet de manipuler des données qui ne sont pas disponibles par défaut dans le langage C (type booléen, type string, tables de hachage, listes ...). Elle met à la disposition du programmeur des fonctions permettant de manipuler ces types beaucoup plus facilement.

- La bibliothèque GObject permet essentiellement à des programmeurs C d'orienter le code en objet et d'implémenter des signaux permettant à plusieurs objets de communiquer entre eux.

Vala va générer du code C s'appuyant essentiellement sur ces deux bibliothèques.

## Approche adoptée dans ce tutoriel

La première partie traitera de l'installation de vala et de la mise en place de l'environnement de programmation.

La deuxième partie traitera des bases du langage et se limitera à une approche impérative (ou procédurale). On y abordera la manipulation des types de base, l'apprentissage des structures de contrôle (condition et boucle) et l'apprentissage des fonctions. Ces deux premières parties sont surtout destinées aux débutants, qui abordent pour la première fois un langage de programmation.

Une troisième partie traitera de la programmation objet en Vala et des types évolués (tableau, liste, liste associative). Une quatrième partie fera un rapide tour d'horizon de quelques bibliothèques disponibles sous Vala (Gtk+, Sqlite, Cairo). Leurs utilisations sera illustré par des exemples.

Tous les exemples de ce tutoriel ont été exécutés sous une distribution GNU-Linux Debian, version Buster. Ils devraient normalement fonctionner de la même manière sous Windows et MacOS-X.

# 2. Installation de Vala

## 1. Installation sous MacOS-X

Il faut tout d'abord installer Homebrew. Homebrew est un gestionnaire de paquet qui regroupe des milliers d'applications libres qui ne sont pas disponibles sur le store d'Apple.

La publication sur le store d'Apple est payante et ceci s'applique également pour les logiciels libres. Seul des fondations ou des entreprises peuvent donc placer leurs logiciels dans le Store d'Apple.

L'ensemble de la plus part des logiciels libres sous MacOS-X est regroupé dans Homebrew.

Ouvrez un terminal (taper **terminal** dans le Finder) et entrez la commande suivante:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Entrez ensuite votre mot de passe.

Pour installer vala tapez ensuite :

```
brew install vala
```

Vous aurez également besoin d'un environnement de développement. Celui maintenu par Apple, X-Code, ne prend pas en charge la coloration syntaxique de Vala. Nous vous conseillons d'installer et d'utiliser un de ces deux environnements : anjuta ou gnome-builder.

Ils prennent en charge la coloration syntaxique et la complétion de code.

Ils sont tous les deux disponibles en utilisant Homebrew. Par exemple, si vous voulez installer gnome-builder, ouvrez un terminal et tapez :

```
brew install gnome-builder
```

Dans ce tutoriel nous utiliserons **gnome-builder** mais vous pouvez utiliser l'éditeur que vous voulez.

## 2. Installation sous Linux

Cela dépendra de votre distribution et du gestionnaire de paquet (apt, rpm, pacman ...). Par exemple sous Debian, connectez-vous en utilisateur root et taper la commande suivante :

```
apt-get install valac
```

## 3. Installation sous Windows

Vous pouvez compiler du vala sous Windows en utilisant MSYS2 à télécharger à l'adresse suivante : <http://www.msys2.org/>.

Cet environnement utilise une version améliorée de **mingw**, logiciel permettant de simuler un environnement de type Unix. MSYS2 utilise **pacman** comme gestionnaire de paquet. Une fois le téléchargement effectué, lancez l'exécutable **msys2.exe**. Un terminal devrait s'ouvrir. On va d'abord synchroniser le dépôt en tapant :

```
pacman -Syu
```

Cette commande est nécessaire pour récupérer une mise-à-jours des dépôts logiciels. Télécharger ensuite les paquets nécessaires en tapant les commandes suivantes :

```
pacman -S mingw-w64-x86_64-gcc
pacman -S mingw-w64-x86_64-pkg-config
pacman -S mingw-w64-x86_64-vala
```

Si vous cherchez un paquet contenant une expression particulière taper : **pacman -Ss expression**

Par exemple :

```
pacman -Ss geany
```

Puis, pour installer geany :

```
pacman -S mingw-w64-x86_64-geany
```

# 3. Premier pas

## 1. Introduction

### 1. Utilitaires pour programmer

Que ce soit sous Linux, Windows ou MacOS-X nous allons utiliser une méthode commune pour programmer. Ainsi tous les exemples de ce tutoriel pourront être exécutés à peu près de la même manière sur n'importe quel système d'exploitation.

Cette approche aura également un avantage non négligeable. En compilant de la même manière vos applications vala à la fois sur Linux, Windows et MacOS-X, vous réglerez la plus part des problèmes de portabilité et votre application fonctionnera sur tous les systèmes d'exploitation.

Nous utiliserons un terminal à la fois pour créer les répertoires de nos programmes, pour lancer l'éditeur de texte (ou l'EDI) puis pour compiler. Nous allons détailler dans ce chapitre une manière de faire. À charge pour le lecteur de reproduire ou d'adapter cette manière dans tous les exemples qui suivront. Nous prendrons comme éditeur de texte **gnome-builder**. Celui-ci est disponible sur les trois systèmes d'exploitation, il est simple d'utilisation, prend en charge la coloration syntaxique et la complétion de code. Mais rien ne vous empêche d'utiliser un autre éditeur.

### 2. Ouverture d'un terminal

Le terminal permet de dialoguer avec le système d'exploitation par le biais d'un langage appelé **le shell**. Les possibilités d'un shell sont immenses mais dans ce tutoriel nous n'utiliserons que les commandes nécessaires à la programmation.

## 1 Sous MacOS-X

MacOS-X est basé sur un Unix-like (couche BSD par dessus un micro-noyau GNU-Mach). Un terminal est donc disponible par défaut. Taper simplement le mot **terminal** dans le Finder

## 2 Sous Windows

Démarrer l'exécutable MSYS2, un terminal s'ouvre mais celui-ci servira essentiellement à gérer les dépôts logiciels et à installer de nouveaux logiciels. Un exécutable « mingw64.exe » se trouve à la racine. Taper dans le terminal en cours :

```
cd/
```

Taper ensuite la commande :

```
./mingw64.exe
```

Un deuxième terminal s'ouvre. C'est dans celui-ci que vous travaillerez et que vous devrez taper les commandes des chapitres suivants.

## 3 Sous Linux

Cela dépend de votre environnement de bureau. Par exemple sous le bureau Mate vous trouverez les terminaux dans le menu **Outils systèmes**.

### 3. Les commandes de base du shell

Pour plus de détails sur les commandes du shell, vous pouvez vous reporter à l'Annexe. Nous allons nous contenter dans ce chapitre des commandes élémentaires permettant de créer des répertoires, de lister leur contenu et de s'y déplacer. Nous verrons ensuite les commandes permettant de compiler des fichiers sources afin de créer des binaires et la manière de les exécuter.

## 4 Connaître le répertoire courant

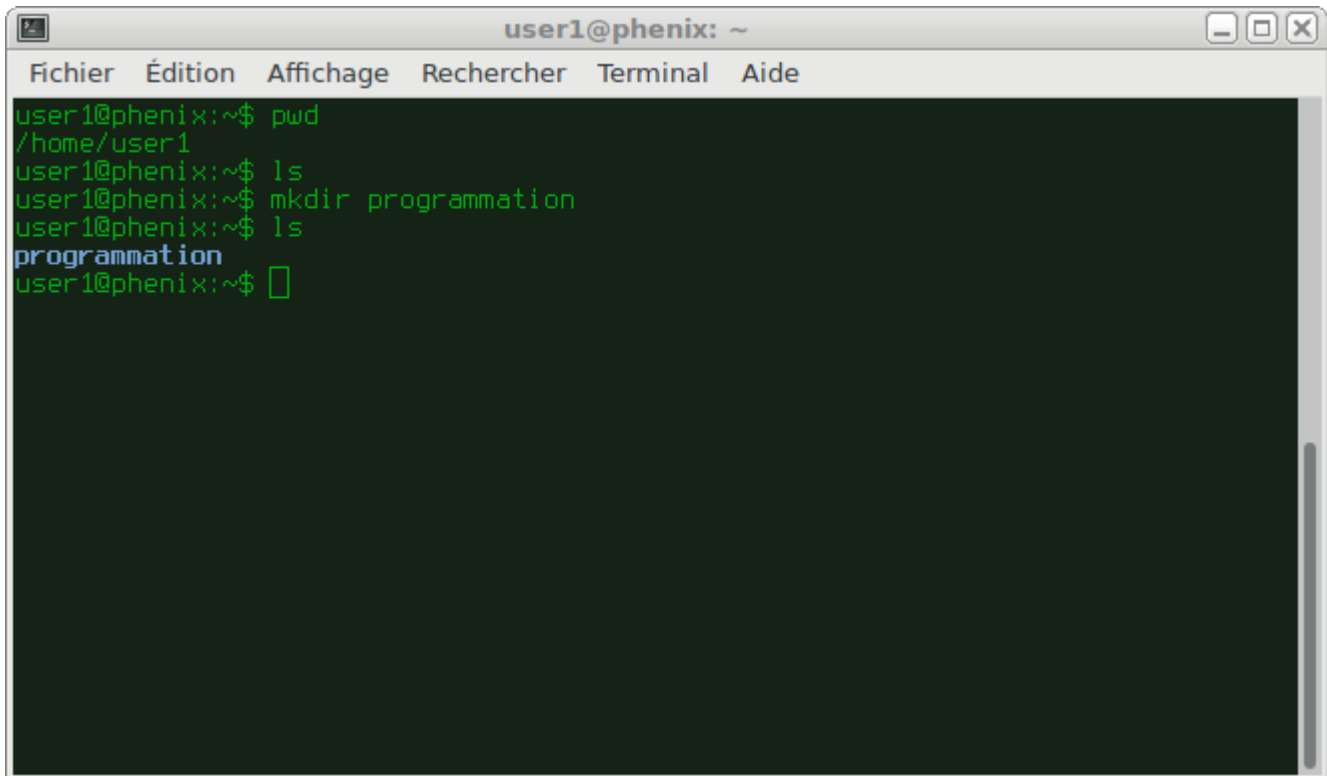
Le répertoire courant est celui dans lequel nous nous trouvons. La commande **pwd** vous donnera le chemin qui amène au répertoire courant. Tous les répertoire descendent du répertoire racine dont le symbole est le slash (/).

## 5 Création d'un répertoire de travail et affichage du contenu

Nous allons tout d'abord créer un répertoire de travail dans lequel nous enregistrerons et exécuterons les exemples de ce tutoriel. Nous appellerons ce répertoire de travail **programmation**. Dans le terminal taper la commande :

```
mkdir programmation
```

Vous pouvez vous assurer de la création de ce répertoire avec la commande **ls** qui devrait afficher le contenu du répertoire courant. La capture d'écran qui suit résume ces différentes étapes :



```
user1@phenix: ~  
Fichier  Édition  Affichage  Rechercher  Terminal  Aide  
user1@phenix:~$ pwd  
/home/user1  
user1@phenix:~$ ls  
user1@phenix:~$ mkdir programmation  
user1@phenix:~$ ls  
programmation  
user1@phenix:~$
```

Dans l'exemple ci-dessus, la commande **pwd** renvoi */home/user1*. On se situe donc dans ce répertoire. La commande **ls** ne retourne rien donc le contenu de */home/user1* ne contient rien (ni fichier, ni répertoire).

On crée un répertoire nommé *programmation* avec la commande **mkdir**. On affiche de nouveau le contenu avec **ls** et on constate que le répertoire a bien été créé.

## 6 Se déplacer dans les répertoires

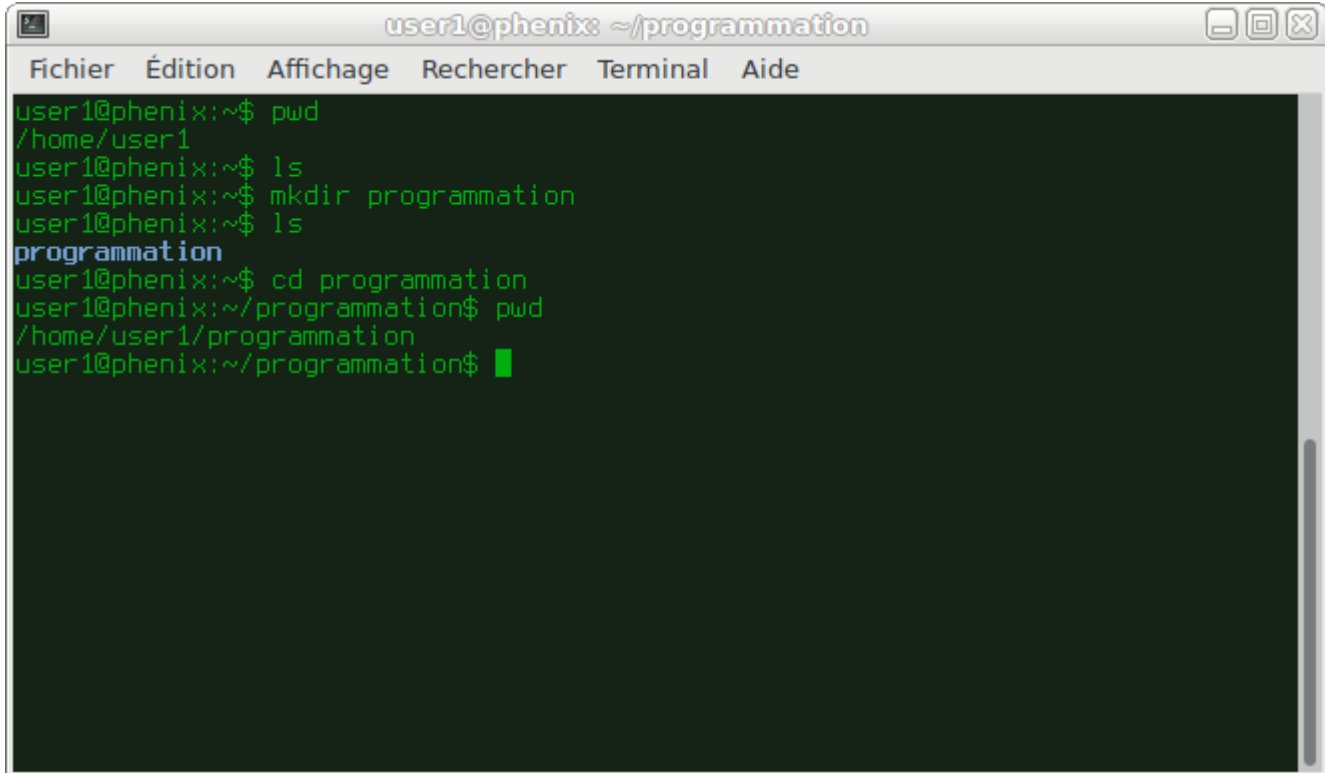
Pour se déplacer dans les répertoires on utilise la commande **cd**. Taper dans le terminal la commande :

```
cd programmation
```

*Si vous voulez plus de détails sur la commande **cd** et la gestion des répertoires vous pouvez consulter l'annexe.*

Vous pouvez vérifier que vous êtes dans le bon répertoire en utilisant de nouveau la commande **pwd**.

La capture d'écran suivante résume l'ensemble de toutes les commandes précédentes :



```
user1@phenix: ~/programmation
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
user1@phenix:~$ pwd
/home/user1
user1@phenix:~$ ls
user1@phenix:~$ mkdir programmation
user1@phenix:~$ ls
programmation
user1@phenix:~$ cd programmation
user1@phenix:~/programmation$ pwd
/home/user1/programmation
user1@phenix:~/programmation$ █
```

Nous testerons les exemples de ce tutoriel dans le répertoire **programmation**. Il faudra donc, à chaque démarrage du terminal vous assurez que vous vous situez bien dans ce répertoire (en vous aidant des commandes **cd** et **pwd**). Afin de mieux nous organiser, nous allons créer des sous-répertoires correspondants aux différents exemples de ce tutoriel. Nous allons commencer par créer un répertoire **premier\_pas** et nous allons nous placer à l'intérieur de celui-ci :

```
mkdir premier_pas
cd premier_pas
```

## 7 Création de notre fichier source et lancement de l'éditeur de texte

Nous allons créer un fichier dans lequel nous écrirons du code *vala* puis nous allons ouvrir ce fichier avec un éditeur de texte. Pour créer un fichier vide nous utilisons la commande **touch**. Nous appellerons ce fichier **exemple.vala**. Dans le terminal il faudra taper la commande suivante :

```
touch exemple.vala
```

Vous pouvez vérifier que le fichier a bien été créé avec la commande **ls**.

Il ne manque plus qu'à lancer notre éditeur de texte. Dans l'exemple suivant, on utilise *gnome-builder*, qui sera suivi du fichier à éditer :

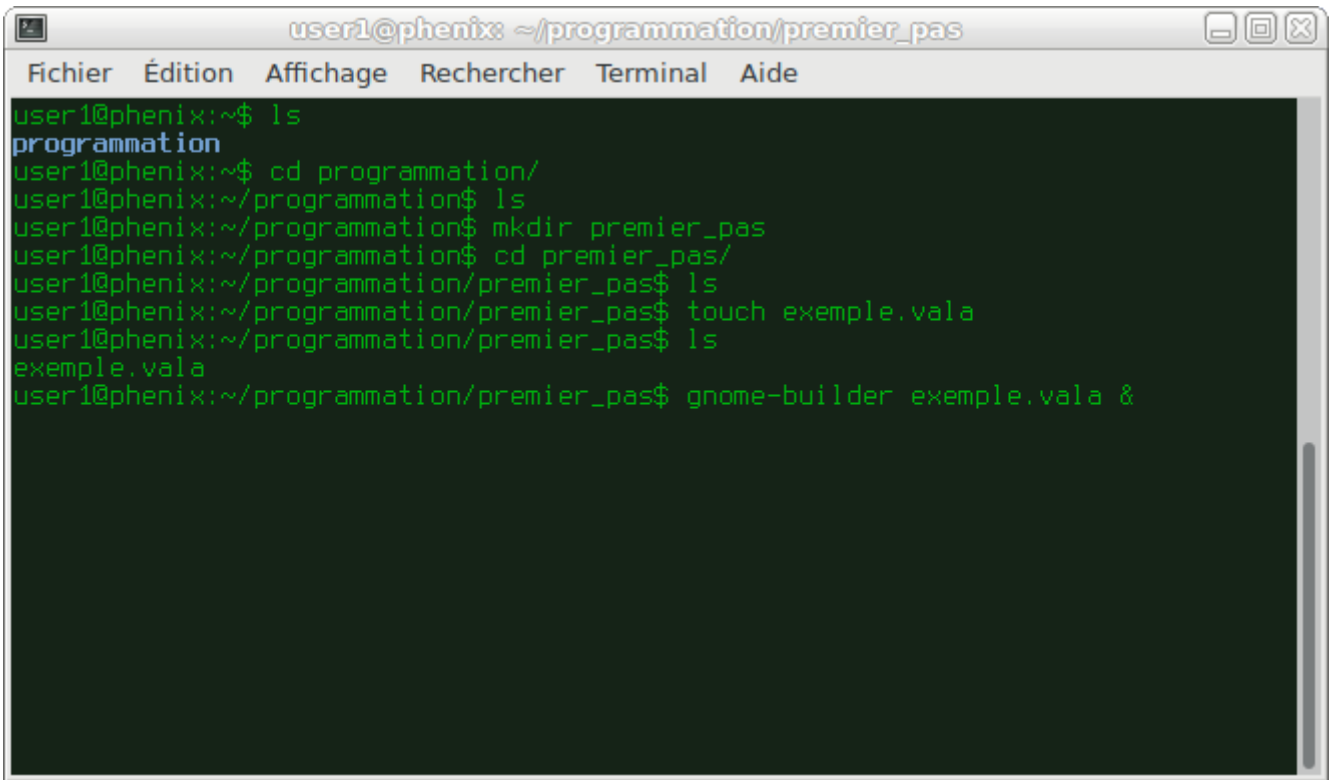
```
gnome-builder exemple.vala &
```

Le symbole **&** (touche Shift + 1) à la fin de la commande permet de reprendre la main dans le terminal. Ce qui nous sera utile pour lancer les commandes de compilation. Si vous omettez ce symbole, l'éditeur de texte mobilisera le terminal et vous serez obligé de fermer l'éditeur pour reprendre la main.

On peut noter également que *gnome-builder* dispose d'un terminal intégré. On pourrait donc directement compiler notre programme en utilisant le terminal de **gnome-builder**.

La capture d'écran suivante récapitule l'ensemble des commandes précédentes :





```
user1@phenix: ~/programmation/premier_pas
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
user1@phenix:~$ ls
programmation
user1@phenix:~$ cd programmation/
user1@phenix:~/programmation$ ls
user1@phenix:~/programmation$ mkdir premier_pas
user1@phenix:~/programmation$ cd premier_pas/
user1@phenix:~/programmation/premier_pas$ ls
user1@phenix:~/programmation/premier_pas$ touch exemple.vala
user1@phenix:~/programmation/premier_pas$ ls
exemple.vala
user1@phenix:~/programmation/premier_pas$ gnome-builder exemple.vala &
```

L'éditeur **gnome-builder** devrait alors s'ouvrir.

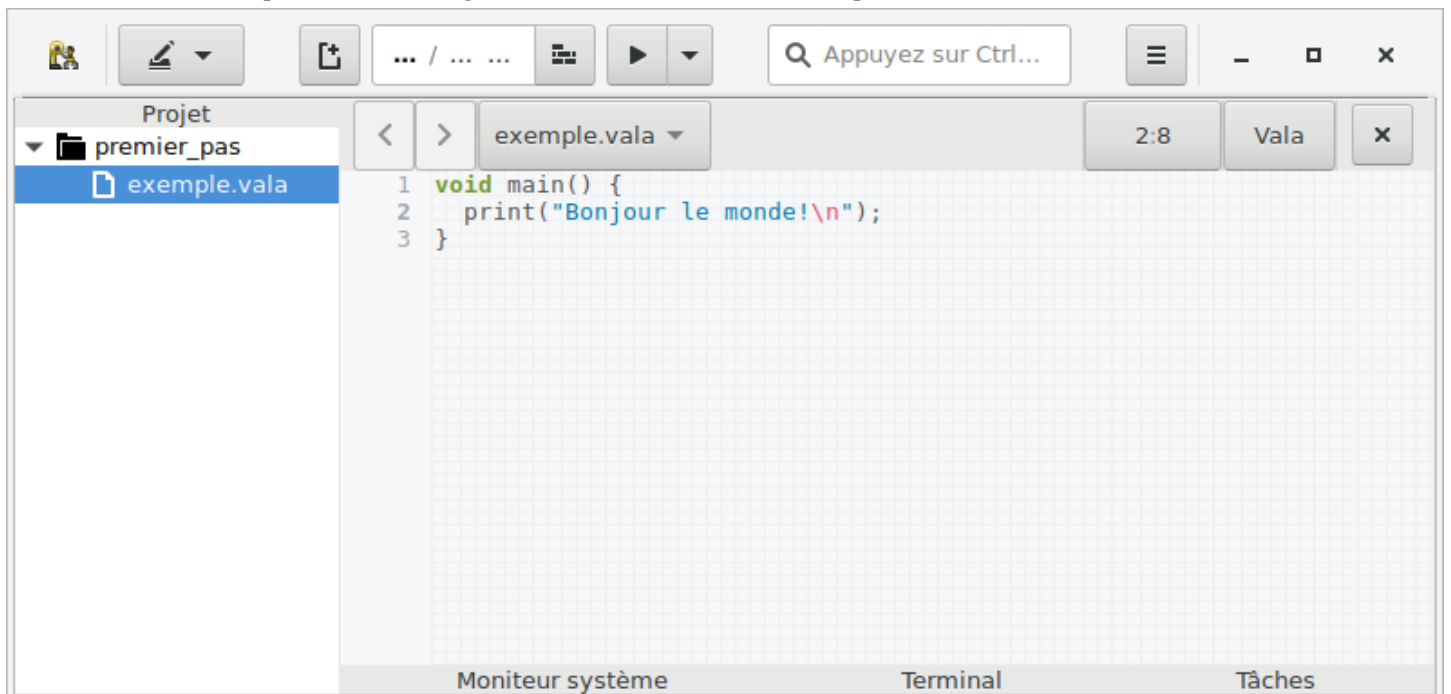
## 8 Premier exemple de programme

Une fois l'éditeur ouvert nous allons taper le programme suivant :

```
void main(){
    print("Bonjour le monde!\n");
}
```

Les accolades ouvrantes et fermantes sont obtenus par la combinaison des touches **Alt Gr + 4** et **Alt Gr + =**

Ci-dessous, une capture d'écran de **gnome-builder** contenant le code précédent :



```
Projet
└─ premier_pas
   └─ exemple.vala
      1 void main() {
      2     print("Bonjour le monde!\n");
      3 }
```

Il faut ensuite enregistrer les modifications, soit en utilisant le menu et en cliquant sur **Enregistrer tout** ou soit en utilisant la combinaison des touches **Ctrl+S**.

L'ordinateur ne comprend que le binaire (suite de 0 et de 1). Il faut donc traduire ce fichier source en langage binaire. On utilise pour cela un logiciel appelé compilateur. Pour compiler du code vala on utilise le compilateur **valac**.

## 9 Compiler notre programme

On retourne dans le terminal à partir duquel on a lancé `gnome-builder`. Puis on tape la commande suivante :

```
valac exemple.vala
```

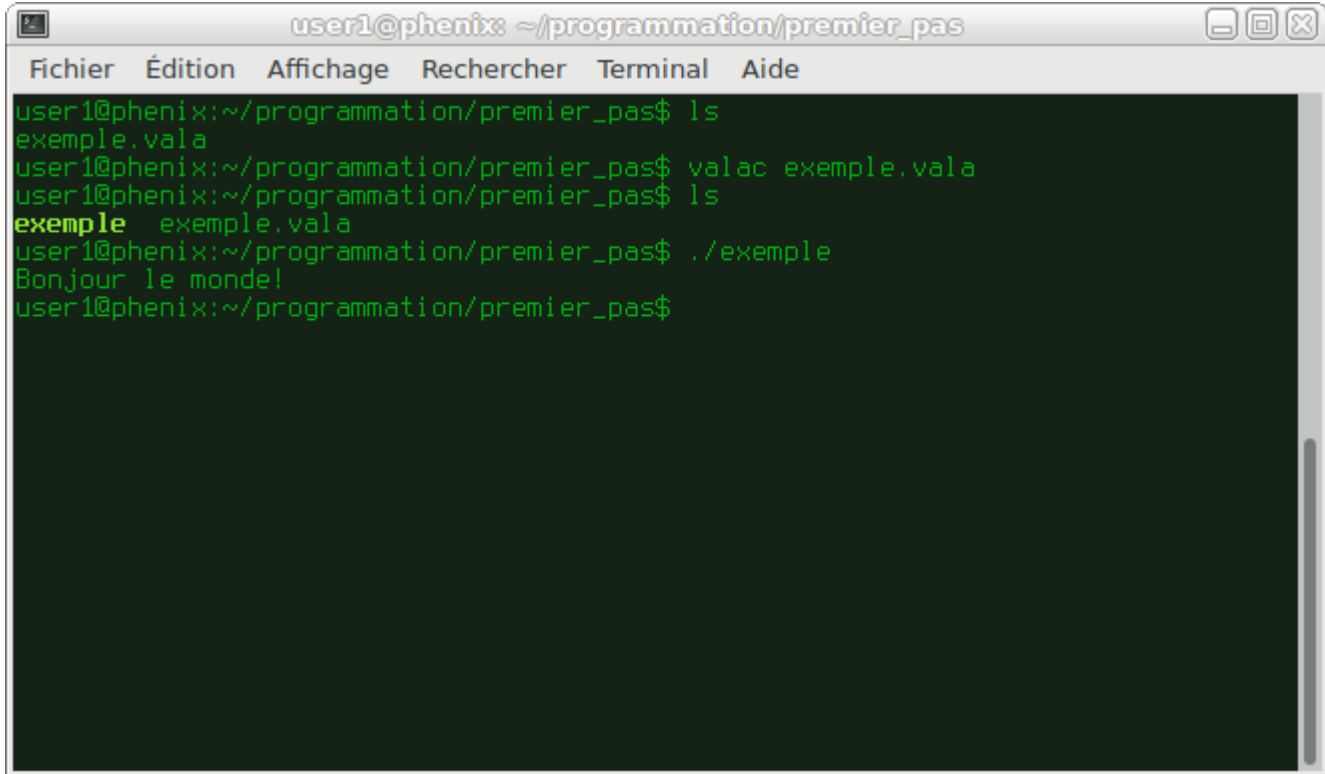
En absence de message d'erreur, le compilateur devrait créer un exécutable nommé **exemple**. Vous pouvez vous en assurer avec la commande `ls`.

## 10 Lancer l'exécutable

Pour exécuter :

```
./exemple
```

Vous devriez voir le message « Bonjour le monde ! » s'afficher dans le terminal comme le montre la capture ci-dessous :



```
user1@phenix: ~/programmation/premier_pas
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
user1@phenix:~/programmation/premier_pas$ ls
exemple.vala
user1@phenix:~/programmation/premier_pas$ valac exemple.vala
user1@phenix:~/programmation/premier_pas$ ls
exemple  exemple.vala
user1@phenix:~/programmation/premier_pas$ ./exemple
Bonjour le monde!
user1@phenix:~/programmation/premier_pas$
```

## 11 Combiner plusieurs lignes de commande en une seule

On peut combiner plusieurs commandes shell en les séparant par un point virgule. Le symbole **&&** permet d'enchaîner une commande à condition que la précédente ait bien été exécuté. La commande suivante va donc effacer l'écran grâce à la commande `clear`, puis compiler le programme et enfin l'exécuter dans le cas où la compilation a réussi.

```
clear; valac exemple.vala && ./vala
```

### 2. Quelques explications sur notre premier programme

Un programme informatique est composé de plusieurs fonctions qui sont chargées d'exécuter des instructions. Il y a par exemple une fonction servant à afficher du texte (fonction **print**), une autre permettant de lire des valeurs entrées au clavier, d'autres permettant de convertir des nombres entiers en chaîne de caractère et ainsi de suite.

Le programmeur peut lui-même élaborer ses propres fonctions mais ce chapitre sera abordé ultérieurement. En Vala il existe une fonction particulière appelée `main`. Si un programme comporte plusieurs fonctions, c'est la fonction **main()** qui sera toujours appelé en premier. Une fonction Vala est de la forme :

```
Type_fonction nom_fonction (arguments_fonction) {
    Code de la fonction ...
    .....
}
```

Les accolades ouvrantes et fermantes délimitent un **bloc d'instruction**. Elles sont obtenues par la combinaison des touches **Alt Gr + 4** et **Alt Gr + =**

Ici la seule instruction du bloc est la fonction **print()** dont le rôle est d'afficher du texte.

Une fonction peut renvoyer des valeurs de tous types (un nombre entier, un nombre à virgule, une chaîne de caractère ...). Lorsqu'elle ne renvoie aucun type particulier, on utilise le mot clé **void** devant la déclaration de la fonction. Nous verrons la signification des arguments plus tard, dans le chapitre sur les fonctions. Dans un premier temps nous utiliserons des parenthèses vides et nous mettrons systématiquement le mot clé **void** en début du main en guise de simplification.

Il faut retenir qu'une fonction comporte toujours des parenthèses, même si le contenu est vide.

### 3. Afficher du texte avec la fonction print

La fonction **print()** affiche du texte dans l'écran d'un terminal. Pour formater le texte on peut utiliser des séquences d'échappement.

#### 1. Séquences d'échappement les plus courant

Symbole	Signification
\n	Retour de ligne
\r	Efface jusqu'au début de la ligne
\t	Tabulation horizontale
\v	Tabulation verticale
\\	Affiche un antislash (\)
\'	Affiche un simple quote
\"	Affiche un guillemet double

#### Exemple de programme

Fichier : print.vala

```
void main() {
    print("Bonjour le monde!\n");
    print("Bonjour \t le monde!\n");
}
```

Compilation et exécution :

```
valac print.vala && ./print
```

Résultat en sortie :

```
Bonjour le monde!
Bonjour      le monde!
```

Dans les exemples suivants, nous utiliserons souvent l'échappement **\n** correspondant au retour de ligne. Nous les mettrons systématiquement en fin de chaîne afin d'avoir plus de visibilité dans l'affichage.

#### 2. Spécificateur de format

Lorsque l'on veut afficher des valeurs entières on utilise le spécificateur **%i** ou **%d**.

Fichier : testPrint.vala

```
void main() {
    print("Ceci est le nombre %i\n", 4);
    print("Voici deux nombres qui se suivent : %i et %i\n", 3,4);
}
```

Compilation et exécution :

```
valac testPrint.vala && ./testPrint
```

Résultat en sortie :

```
Ceci est le nombre 4
Voici deux nombres qui se suivent : 3 et 4
```

Le premier **%i** est remplacé par le premier nombre après la virgule (c'est-à-dire le chiffre 3) et le deuxième **%i** est remplacé par le deuxième nombre (et ainsi de suite).

L'exemple ci-dessus ne présente pas beaucoup d'intérêt car on aurait pu directement remplacer les **%i** par les nombres correspondants à l'intérieur des guillemets. Cependant cette technique permet par exemple de faire des calculs sur des nombres rapidement et d'afficher le résultat à l'intérieur d'une phrase. Par exemple :

```
print("Somme de 3 et 4:%i\n", 3+4);
```

## 4. Lire une valeur entrée au clavier avec `read_line` - premier aperçu d'une variable

Pour lire du texte ou une valeur entrée au clavier, on utilise la fonction `read_line`. La fonction `read_line` appartient à la bibliothèque `stdin` (abréviation de standard input). On devra donc précéder la fonction par `stdin` suivi d'un point (`stdin.read_line()`) → ne pas oublier de mettre des parenthèses à la fin d'une fonction).

La fonction `read_line()` renvoie une valeur de type « chaîne de caractère », qui correspond au mot clé `string` en vala. La valeur entrée sera stockée en mémoire dans une variable qui sera précédé de son type :

```
string nom_variable = stdin.read_line();
```

Dans l'exemple précédent nous avons vu que pour afficher un nombre entier, on utilisait le spécificateur `%i` ou `%d`. Pour afficher une chaîne de caractère on utilisera le spécificateur `%s`.

Fichier : testRead.vala

```
void main() {
    print("Entrez votre nom : ");
    string nom=stdin.read_line();
    print("Nom : %s\n", nom);
}
```

Compilation et exécution :

```
valac testRead.vala && ./testVala
```

Résultat en sortie :

```
Entrez votre nom : Alexandre
Nom : Alexandre
```

## 4. Les types simples

Vala est un langage typé. Cela signifie qu'avant d'effectuer des opérations sur une variable, il faudra d'abord la déclarer en précisant de quel type de variable il s'agit (valeur entière, nombre à virgule, variable de type caractère ...). Il existe une multitude de type différent en Vala. Il est quasiment impossible de tous les énumérer en détail et il sera souvent nécessaire de se référer à la documentation officielle. Néanmoins, dans ce chapitre, nous allons passer en revue les types de base qui sont les plus utilisés. La plus part de ces types proviennent du C et de son extension issue de la glib. Les types plus élaborés, comme les structures, les listes, les tableaux et bien d'autres seront traités dans un autre chapitre.

### 1. Le type `int`

Le type `int` est utilisé pour des variables entières (nombre sans virgules). On déclare une variable de type entière en précédant le nom de la variable par `int`. L'exemple suivant déclare une variable nommé `a` :

```
int a;
```

On pourrait donner un nom de variable plus explicite en fonction du problème posé :

```
int age;
```

(*Attention ! Ne jamais utiliser d'accent dans le nom d'une variable*)

On peut initialiser une variable directement lors de sa déclaration :

```
int a=3;
```

### 1. Afficher un entier

On peut afficher un entier en utilisant les fonctions `stdout.printf` ou `print`. On utilise le spécificateur de format `%i`.

Entrez le contenu ci-après dans un fichier que vous nommerez testInt.vala

Fichier : testInt.vala

```
void main() {
    int a=20;
    int b,c;

    print("a = %i\n", a);
    print("a*a = %i\n", a*a);
    b=a+2;
    c=b*2;
    print("b = %i \t c= %i \n", b, c);
}
```

Compilation :

```
valac testInt.vala
```

Exécution :

```
./testInt
```

Résultats en sortie :

```
a = 20
a*a = 400
b = 22      c= 44
```

## 2. Conversion d'un type int vers un type string

Pour convertir un type entier vers une chaîne de caractère on utilise la méthode `to_string()`. Cette méthode provient de la glib.

```
void main() {
    int a=20;
    print("Nombre %i\n", a);
    print("Nombre %s\n", a.to_string());

    int j=3;
    string phrase = "Il y a " + j.to_string() + " jours de cela\n";
    print(phrase);
}
```

Résultats en sortie :

```
Nombre 20
Nombre 20
Il y a 3 jours de cela
```

## 2. Le type booléen (bool)

Le type booléen peut prendre deux valeurs. Soit vraie (true), soit faux (false).

### 1. Afficher un type booléen

Pour afficher une variable booléenne on fait précéder le nom de la variable par le symbole `$`. La chaîne de caractère est toujours comprise entre des guillemets mais elle sera précédé par le symbole `@`.

Taper le code suivant dans un fichier que vous nommerez `test_bool.vala`

```
void main() {
    bool test=true;
    print(@"valeur de la variable : $test\n");
}
```

Compilez puis exécutez en tapant la commande suivante :

```
valac test_bool.vala && ./test_bool
```

Vous obtenez la sortie suivante :

```
valeur de la variable : true
```

### 2. Conversion d'un type booléen vers un type string

Comme pour les entiers, on utilise la méthode `to_string()`.

### 3. Conversion d'un type string vers un type booléen

On utilise la méthode `parse` :

```
string t = "true";
bool tb = bool.parse(t);
```

## 3. Le type char

Ce type correspond à un seul caractère. Dans l'exemple ci-dessous on déclare une variable de type `string` qui contient la phrase **bonjour le monde**. On assigne ensuite, dans deux variables de type `char`, la première lettre et la quatrième lettre de la phrase.

La position de la lettre est délimité par le crochet ouvert [ (AltGr+5) et le crochet fermé ] (AltGr+)]. On remarque que la première position commence par 0 et pas par 1.

Fichier : testChar.vala

```
void main() {
    string phrase = "bonjour le monde!";
```

```

char lettre1=phrase[0];
char lettre2=phrase[3];
print("Lettre1=%c et Lettre2=%c\n", lettre1, lettre2);
}

```

Compilation et exécution :

```
valac testChar.vala && ./testChar
```

Résultat :

```
Lettre1=b et Lettre2=j
```

## 2. Exemple - obtenir le caractère le plus grand

On utilise la méthode `max` pour obtenir le caractère le plus grand.

Fichier : maxChar.vala

```

void main () {
    string sa;
    string sx;
    char ca;
    char cx;
    print("choisissez une premiere lettre :\n");
    sa = stdin.read_line();
    print("choisissez une seconde lettre :\n");
    sx = stdin.read_line();
    ca = sa[0];
    cx = sx[0];
    char maxChar = char.max ( ca, cx);
    print ("maximum = %c\n", maxChar);
}

```

Compilation et exécution :

```
./maxChar.vala && ./maxChar
```

Résultat :

```

choisissez une premiere lettre :
h
choisissez une seconde lettre :
e
maximum = h

```

## 4. Le type double

Le type **double** permet de manipuler les nombres à virgules.

## 5. Le type string

### 1. Concaténer des chaînes de caractères

Le type `string` permet de manipuler les chaînes de caractères. Pour concaténer plusieurs chaînes on peut utiliser l'opérateur d'addition :

```

void main () {
    string s1 = "bonjour";
    string s2 = " le ";
    string s3 = "monde!";
    print(s1+s2+s3+"\n");
}

```

### 2. Conversion d'un type string vers un type int

Pour convertir une chaîne de caractère en un entier, on utilise la méthode `int.parse()`.

```
int number=int.parse("4321");
```

### 3. Variable string déclaré sur plusieurs lignes

Pour assigner une variable de type `string` sur plusieurs ligne on peut utiliser les triples guillemets :

```

string requete = """
SELECT * FROM MA_TABLE WHERE

```

```
CLIENT='DUPONT'  
"";
```

## 5. Les structures de contrôle

Il existe deux types de structure de contrôle : les conditions et les boucles.

Les conditions permettent de comparer des valeurs entre elles et les boucles d'effectuer des opérations répétitives. Les structures de contrôles sont à la base de tous les programmes informatique.

### 1. Les conditions

#### 1. Instruction if

Dans l'exemple suivant on génère un nombre aléatoire entre 1 et 100. On enregistre le résultat dans la variable **val**. On va ensuite comparer cette valeur par rapport au nombre 50.

Fichier testIf.vala :

```
void main(){  
    int val = GLib.Random.int_range(1, 100);  
    if(val > 50){  
        print("la valeur est au dessus de 50 et est : %d \n", val);  
    }  
    else{  
        print("la valeur est en dessous de 50 et est : %d \n", val);  
    }  
}
```

Compilation et exécution :

```
valac testIf.vala && ./testIf
```

#### 2. Instruction switch/case

### 2. Les boucles

#### 1. Boucle for

Fichier testFor.vala :

```
void main() {  
    for (int i=0; i < 10; i++) {  
        print("Boucle %d\n", i);  
    }  
}
```

Compilation et exécution :

```
valac testFor.vala && ./testFor
```

Résultat en sortie :

```
Boucle 0  
Boucle 1  
Boucle 2  
Boucle 3  
Boucle 4  
Boucle 5  
Boucle 6  
Boucle 7  
Boucle 8  
Boucle 9
```

#### 2. Boucle foreach

Une boucle foreach permet de parcourir les éléments d'une liste ou d'un tableau.

Dans l'exemple qui suit nous déclarons un tableau contenant les nombres allant de 1 à 5 et nous affichons ensuite les valeurs de ce tableau :

```
void main() {
    int[] tab={1,2,3,4,5};
    foreach (int val in tab) {
        print("val=%d\n", val);
    }
}
```

## 4. Boucle while

## 5. Boucle do ... while

# 6. Les fonctions

## 1. Introduction

Jusqu'à présent nous avons travaillé avec une seule fonction, celle qui correspond au **main()**. Les fonctions servent à effectuer différentes opérations de manières séparées. Imaginons un programme qui demande à l'utilisateur de rentrer deux valeurs entières et qui donne en sortie l'addition de ces deux nombres. Un premier exemple (sans utiliser d'autres fonctions que le main) serait donné par le programme suivant :

Fichier : addition.vala

```
void main() {
    string a, b;
    print("Donnez un premier nombre : ");
    a = stdin.read_line();
    print("Donnez un deuxieme nombre : ");
    b = stdin.read_line();
    print("a+b=%i\n", int.parse(a) + int.parse(b));
}
```

Exécution est compilation :

```
valac addition.vala && ./addition
```

Résultat :

```
Donnez un premier nombre : 45
Donnez un deuxieme nombre : 21
a+b=66
```

Imaginons maintenant un programme, qui permet à l'utilisateur d'avoir le choix d'effectuer soit l'addition ou soit la multiplication de ces deux nombres.

Fichier : additionMultiplication.vala

```
int addition(int a, int b) {
    return a+b;
}

int multiplication(int a, int b) {
    return a*b;
}

void main() {
    string choix, a, b;
    int resultat=0;
    print("Donnez un premier nombre : ");
    a = stdin.read_line();
    print("Donnez un second nombre : ");
    b = stdin.read_line();
    print("Entrez 1 pour additionner deux nombres ou 2 pour les multiplier : ");
    choix = stdin.read_line();
    if (int.parse(choix) == 1) resultat = addition(int.parse(a), int.parse(b));
    if (int.parse(choix) == 2) resultat = multiplication(int.parse(a), int.parse(b));
}
```



```
print("Resultat : %d\n", resultat);
}
```

## 7. Programmation objet

### 1. Les mutateurs (set)

```
public class App {
    private string name;

    public void set_name(string _name) {
        this.name=_name;
    }

    public static int main() {
        App app=new App();
        app.set_name("Alexandre");
        stdout.printf("Nom : %s\n", app.name);
        return 0;
    }
}
```

### 2. Les accesseurs (get)

## 8. Les types évolués

### 1. Le type array

#### 1. Tableau à une dimension

##### 1. Déclaration d'un tableau à une dimension

On peut déclarer un tableau en utilisant l'opérateur **new**. On appelle cela l'allocation dynamique. Dans l'exemple ci-dessous, on déclare un tableau de cinq entiers :

```
int[] a = new int[5];
```

On peut initialiser un tableau lors de sa création. Dans l'exemple suivant on initialise un tableau d'entier comportant trois éléments :

```
int[] a={2, 5, 8};
```

On utilise souvent une variable de type constante afin de fixer la taille du tableau en début de programme. Cela permet d'utiliser le nom de cette constante dans la suite du programme (boucle **for** et **foreach** par exemple) et de pouvoir modifier la taille du tableau facilement sans retoucher plusieurs portions de code. Dans ce cas nous sommes obligés d'utiliser l'allocation dynamique de la manière suivante :

```
const int SIZE_ARRAY[10];
int [] my_array = new int[SIZE_ARRAY];
```

On peut afficher un élément particulier du tableau en utilisant les crochets. L'exemple suivant va afficher le premier élément du tableau (on commence par zéro et pas par un) :

```
print("val=%i\n", a[0]);
```

#### 2. Taille d'un tableau

On obtient la taille d'un tableau en utilisant la méthode **length**.

```
int[] a={3, 7, 14};
int size = a.length;
print("Taille du tableau : %d\n", size);
```

#### 3. Parcours d'un tableau

La méthode **length** permet de parcourir les éléments d'un tableau facilement, comme le montre l'exemple suivant :

```
void display_array(int[] number) {
    for (int i=0; i<number.length;i++) {
        print("%i\n", number[i]);
    }
}
```

```
void main() {
    int[] number = {1, 5, 7};
    display_array(number);
}
```

On pourrait également utiliser l'instruction **foreach** :

```
void main() {
    int[] tab={1,2,3,4,5};
    foreach (int val in tab) {
        print("val=%d\n", val);
    }
}
```

## 2. Copy d'un tableau

On copie facilement un tableau en utilisant l'opérateur d'égalité :

```
int[] aCopy = a;
```

L'exemple suivant illustre la copie d'un tableau :

Fichier copyArray.vala :

```
void afficher_tableau(int[] tab) {
    for (int i=0;i<tab.length;i++) {
        print("%i\n", tab[i]);
    }
}

void main() {
    int[] a={2, 5, 8};
    print("On affiche a[]\n");
    afficher_tableau(a);

    int[] b=a;
    print("\nOn affiche b[]\n");
    afficher_tableau(b);
}
```

Résultat en sortie :

On affiche a[]

```
2
5
8
```

On affiche b[]

```
2
5
8
```

## 3. Tableau multi-dimensionnel

Pour un tableau d'entier à deux dimensions on utilisera la notation :

```
int[,] = new int[2, 4];
```

Pour un tableau d'entier à trois dimensions :

```
int[,,] = new int[2, 3, 5];
```

Un exemple est illustré ci-dessous avec un tableau à deux dimensions de type **string** :

```
void display_array(string[,] aStr) {
    for (int i=0; i<aStr.length[0]; i++) {
        print("%s \t %s \t %s\n", aStr[i, 0], aStr[i, 1], aStr[i, 2]);
    }
}

void main() {
    string[,] mStr = new string[2, 3];
    mStr = {
        {"id", "Nom", "Prenom"},
        {"5", "alex", "durand"}};
}
```

```

        display_array(mStr);

        print("\nAprès modification\n");
        mStr[1, 0] = "7";
        mStr[1, 1]= "toto";
        mStr[1, 2]= "dupond";
        display_array(mStr);
    }

```

Résultat à l'affichage :

id	Nom	Prenom
5	alex	durand

Après modification

id	Nom	Prenom
7	toto	dupond

## 2. Le type enum

Exemple de programme :

```

enum Periode {
    MENSUELLE,
    TRIMESTRIELLE,
    SEMESTRIELLE
}

void message(Periode p) {
    switch (p) {
        case Periode.MENSUELLE:
            print("C'est souvent\n");
            break;
        case Periode.TRIMESTRIELLE:
            print("C'est pas souvent\n");
            break;
        case Periode.SEMESTRIELLE:
            print("C'est rare!\n");
            break;
    }
}

void main() {
    Periode per;
    per = Periode.TRIMESTRIELLE ;
    message(per);
}

```

Résultat en sortie :

C'est pas souvent

## 3. Le type DateTime

### 1. Obtenir la date courante

L'exemple suivant affiche la date courante.

Fichier : dateCurrent.vala

```

void main() {
    var dt = new DateTime.now_local();
    print(dt.format("%x\n"));
    print("%s\n", dt.to_string());
}

```

Compilation et exécution :

```
valac dateCurrent.vala && ./dateCurrent
```

Résultat en sortie (dépend de la date du test) :

```
07/25/19
```

```
2019-07-25T07:53:35+0200
```

## 2. Obtenir le mois courant

L'exemple suivant détermine le mois du jour courant où on fait le test.

Fichier : getMonth.vala

```
void main() {
    var dt = new DateTime.now_local();
    print(dt.format("%x\n"));
    print("Numero du mois : %d\n", dt.get_month());

    switch (dt.get_month()) {
        case 1:
            print("Janvier\n");
            break;
        case 2:
            print("Fevrier\n");
            break;
        case 3:
            print("Mars\n");
            break;
        case 4:
            print("Avril\n");
            break;
        case 5:
            print("Mai\n");
            break;
        case 6:
            print("Juin\n");
            break;
        case 7:
            print("Juillet\n");
            break;
        case 8:
            print("Aout\n");
            break;
        case 9:
            print("Septembre\n");
            break;
        case 10:
            print("Octobre\n");
            break;
        case 11:
            print("Novembre\n");
            break;
        case 12:
            print("Decembre\n");
            break;
    }
}
```

Résultat en sortie (dépend du jour de la date du test) :

```
07/25/19
```

## 4. Le type DateMonth

Le type DateMonth est un cas particulier d'un type énuméré contenant treize valeurs.

Numéro	Valeur correspondantes
0	BAD_MONTH
1	JANUARY
2	FEBRUARY
3	MARCH
4	APRIL
5	MAY
6	JUNE
7	JULY
8	AUGUST
9	SEPTEMBER
10	OCTOBER
11	NOVEMBER
12	DECEMBER

### Exemple de programme

exemple.vala

```
void afficher_mois(DateMonth d) {
    switch (d) {
        case DateMonth.JANUARY:
            print("Janvier\n");
            break;
        case DateMonth.FEBRUARY:
            print("Fevrier\n");
            break;
        case DateMonth.MARCH:
            print("Mars\n");
            break;
        case DateMonth.APRIL:
            print("Avril\n");
            break;
    }
}

void main() {
    DateMonth d;
    d = DateMonth.FEBRUARY;
    afficher_mois(d);

    d = DateMonth.APRIL;
    afficher_mois(d);
}
```

### Compilation et exécution :

```
valac exemple.val && ./exemple
```

### Résultat en sortie :

```
Fevrier
Avril
```

## 1. Comparaison de date

Comme les dates sont classés de 1 à 12, on peut les comparer avec des instructions **if** :

Fichier : exemple2.vala

```
void main() {
```

```

DateMonth d = DateMonth.OCTOBER;
if (d == DateMonth.MAY)
    print("On est en mai\n");
else print("On est pas en mai!\n");
if (d > DateMonth.MAY) print("On est apres le mois de mai\n");

print("\nNouvelle assignation de date:\n");
d = DateMonth.MAY;
if (d == DateMonth.MAY)
    print("On est en mai\n");
else print("On est pas en mai!\n");
if (d > DateMonth.MAY) print("On est apres le mois de mai\n");
}

```

Compilation et exécution :

```
valac exemple2.val && ./exemple2
```

Résultat en sortie :

```

On est pas en mai!
On est apres le mois de mai

```

```

Nouvelle assignation de date:
On est en mai

```

## 5. Le type List

Le type List provient de la glib. On utilise la méthode **append** pour ajouter un élément en fin de liste et la méthode **prepend** pour ajouter un élément en début de liste. On peut parcourir les éléments d'une liste en utilisant l'instruction **foreach**. La déclaration d'un type list se fait de la manière suivante : **List<string>**

### 1. Ajout d'un élément et parcours d'une liste

```

int main (string[] args) {
    List <string> list = new List<string> ();
    list.append("bonjour");
    list.append(" le ");
    list.append("monde!");

    foreach (string str in list) {
        print(str);
    }
    print("\n");
    return 0;
}

```

### 2. Type de retour et passage en argument

Le type de retour d'une fonction d'un type List ou le passage en argument dans une fonction utilise toujours la même notation : **List<string>**

```

List<string> init_name_List() {
    List<string> listStr = new List<string> ();
    listStr.append("Titi");
    listStr.append("Toto");
    listStr.append("Bibi");
    return listStr;
}

void display_List(List<string> listStr) {
    foreach (string name in listStr) {
        print(name+"\n");
    }
}

void main() {
    List<string> my_List = new List<string>();
    my_List = init_name_List();
    display_List(my_List);
}

```

### 3. Retour d'un type List au sein d'une méthode de classe

Le problème va se poser lorsque l'on veut renvoyer un type List lors d'un appel d'une méthode de classe. On devra dans ce cas retourner dans l'accesseur (get) un **copy** de l'objet List de cette manière :

```
return this.list.copy();
```

Cet exemple est illustré ci-dessous :

```
public class TestList {
    private List<string> list;

    public void load_list() {
        this.list.append("voiture");
        this.list.append("camion");
        this.list.append("moto");
    }

    public void display_list() {
        foreach (string item in this.list) {
            print(item+"\n");
        }
    }

    public List<string> get_list() {
        return this.list.copy();
    }

    public TestList() {
        print("Constructeur - Creation de l'objet TestList\n");
    }

    public static int main(string[] args) {
        TestList tl = new TestList();
        tl.load_list();
        tl.display_list();

        print("Duplicata dans une autre liste : ajout d'un element et affichage\n");
        List<string> my_list = tl.get_list();
        my_list.append("velo");

        foreach (string item in my_list) {
            print(item+"\n");
        }

        return 0;
    }
} //end class
```

### 6. Le type HashTable

Le type HashTable provient également de la glib. Il permet de créer des tables associatives de type clé/valeur.

### 7. Le type struct

Contrairement au langage C, une structure peut contenir des champs, des constantes mais aussi des méthodes. Elle peut servir à regrouper des données. On peut par exemple créer une structure « Personne » regroupant plusieurs variables comme le nom, le prénom, l'âge etc ... et créer ses fonctions associées pour modifier et obtenir ses variables.

#### 1. Création et utilisation d'une structure

Cet exemple crée une structure ne contenant que deux données dont une de type entière et une autre de type chaîne de caractère.

Fichier : main.vala

```
struct Data {
    public int nb;
    public string str;
}

int main() {
    Data dat = Data(); // initialisation de la structure

    print ("Valeur par défaut:\n");
    print ("nb=%i \t str=%s \n", dat.nb, dat.str);

    dat.nb=4;
    dat.str="hello!";
}
```

```

        print ("\nValeur apres affectation: \n");
        print ("nb=%i \t str=%s \n", dat.nb, dat.str);
return 0;
}

```

Compilation :

**valac main.vala**

Exécution :

**./main**

Résultat en sortie :

**Valeur par défaut :**

**nb=0 str=(null)**

**Valeur apres affectation :**

**nb=4 str=hello!**

On remarque que par défaut l'instruction **Data dat = Data()** initialise les entiers à 0 et les chaînes de caractères à null.

## 2. Copie de structure

On peut copier facilement une structure dans une autre en utilisant l'opérateur d'égalité.

Fichier : copy\_struct.vala

```

struct Data {
    string surname;
    string firstname;
}

void afficher_data(Data d) {
    print("Nom : %s\n", d.surname);
    print("Prenom : %s\n", d.firstname);
}

void main() {
    Data d = Data();
    d.surname = "Nouvel";
    d.firstname = "Alexandre";
    afficher_data(d);

    Data dCopy = Data();
    dCopy = d;
    afficher_data(dCopy);
}

```

## 3. Tableau de structure

### 1. Tableau statique avec initialisation des champs lors de la création

L'exemple suivant crée un tableau contenant deux structures nommées Data :

Fichier : arrayStaticStruct.vala

```

struct Data {
    public string surname;
    public string name;
}

void afficher_data(Data d) {
    print("-----\n");
    print("Nom : %s\n", d.surname);
    print("Prenom : %s\n", d.name);
}

void main() {
    Data[] listData={ Data(), Data() };
    listData[0].surname = "Nouvel";
    listData[0].name = "Alexandre";
    afficher_data(listData[0]);

    listData[1].surname = "Dupont";
}

```



```
listData[1].name = "Robert";
afficher_data(listData[1]);
}
```

Résultat :

```
-----
Nom : Nouvel
Prenom : Alexandre
-----
```

```
Nom : Dupont
Prenom : Robert
```

On peut faire plus concis en initialisant directement le tableau de structure lors de sa création :

```
Data[] listData={
    Data() {surname="Nouvel", name="Alexandre"},
    Data() {surname="Dupont", name="Robert"}
};
```

## 2. Tableau statique sans initialisation de départ

Parfois le tableau de notre structure évolue au cours du temps et il nous est impossible de l'initialiser lors de sa création. Dans ce cas on déclare le tableau comme un objet en utilisant la directive **new** :

```
struct Data {
    public string surname;
    public string name;
}

void afficher_data(Data d) {
    print("-----\n");
    print("Nom : %s\n", d.surname);
    print("Prenom : %s\n", d.name);
}

void main() {
    Data[] listData = new Data[2];

    listData[0].surname = "Nouvel";
    listData[0].name = "Alexandre";
    afficher_data(listData[0]);

    listData[1].surname = "Dupont";
    listData[1].name = "Robert";
    afficher_data(listData[1]);
}
```

## 3. Fonction retournant un tableau de structure

Une fonction peut renvoyer un tableau de structure en lui donnant le type de la structure suivi des crochets

```
struct Data {
    public string surname;
    public string name;
}

Data[] get_list_data() {
    Data[] ld= {
        Data() {surname="Nouvel", name="Alexandre"},
        Data() {surname="Dupont", name="Robert"}
    };
    return ld;
}

void main() {
    Data[] listData = new Data[2];
    listData = get_list_data();
}
```

```
}
```

#### 4. Tableau de structure comme attribut d'un objet

On peut définir un tableau de structure comme attribut d'un objet. Ce tableau pourra prendre des tailles différentes en fonction des opérations effectuées durant la vie de l'objet.

Fichier: array\_of\_struct.vala

```
/*
 * Compilation & execution
 * clear;valac array_of_struct.vala && ./array_of_struct
 */

public struct Planet{
    string name;
    int ray;
}

public class Test {
    private Planet[] list_planet;

    public void display_list_planet() {
        foreach (Planet p in list_planet) {
            print("name : %s \t ray : %dkm\n", p.name, p.ray);
        }
    }

    public Planet[] new_planet() {
        Planet[] lp = new Planet[3];
        lp[0].name = "Agator";
        lp[0].ray = 5400;
        lp[1].name = "Vanilla";
        lp[1].ray = 7100;
        lp[2].name = "Octaru";
        lp[2].ray = 6200;
        return lp;
    }

    public void change_system() {
        print("Autre systeme solaire\n");
        list_planet = new_planet();
    }

    public Test() {
        Planet[] lp = new Planet[8];
        lp[0].name = "Mercure";
        lp[0].ray = 2440;
        lp[1].name = "Venus";
        lp[1].ray = 6052;
        lp[2].name = "Terre";
        lp[2].ray = 6400;
        lp[3].name = "Mars";
        lp[3].ray = 3390;
        lp[4].name = "Jupiter";
        lp[4].ray = 69900;
        lp[5].name = "Saturne";
        lp[5].ray = 58232;
        lp[6].name = "Uranus";
        lp[6].ray = 25362;
        lp[7].name = "Neptune";
        lp[7].ray = 24622;
        list_planet = lp;
    }

    public static int main(string[] args) {
        Test t = new Test();
        t.display_list_planet();
        t.change_system();
        t.display_list_planet();
        return 0;
    }
}
```

Résultat en sortie :

```

name : Mercure      ray : 2440km
name : Venus       ray : 6052km
name : Terre       ray : 6400km
name : Mars        ray : 3390km
name : Jupiter     ray : 69900km
name : Saturne    ray : 58232km
name : Uranus     ray : 25362km
name : Neptune    ray : 24622km
Autre systeme solaire
name : Agator      ray : 5400km
name : Vanilla    ray : 7100km
name : Octaru     ray : 6200km

```

## 9. Manipulation des répertoires et des fichiers

### Le type File

Le type File de Java provient de la Glib. Pour ouvrir un fichier nous utilisons la méthode **new\_for\_path(string path)**. Par exemple si nous voulons ouvrir un fichier qui s'appelle **data.txt** nous utiliserons l'instruction suivante.

```
File file = File.new_for_path("data.txt");
```

Pour vérifier l'existence d'un fichier nous pouvons utiliser la méthode **query\_exists()** qui renvoie un booléen.

```
File file = File.new_for_path("data.txt");
bool result = file.query_exists();
```

## 10. Séparer le code en plusieurs fichiers – Créer ses propres bibliothèques - Namespaces

### 1.Introduction

Copier le contenu ci-dessous dans le fichier « TestUnit.vala » :

```

namespace TestUnit {
    public void afficher() {
        stdout.printf("TestUnit\n");
    }
} //end namespace TestUnit

```

Puis dans un deuxième fichier que vous nommerez « main.vala » copier le code suivant :

```

using TestUnit;

void main() {
    print("Test\n");
    TestUnit.afficher();
}

```

Pour compiler et exécuter, utiliser la commande suivante :

```
valac main.vala TestUnit.vala &&./main
```

## 11. Étude de quelques bibliothèques

### 1.Manipulation des fichiers avec gio

#### 1. Création et écriture dans un fichier

L'exemple suivant va créer un fichier « testFichier.txt » dans lequel on va écrire la phrase « Une phrase dans le fichier ».

Fichier: createFile.vala

```

void main() {
File file = File.new_for_path("testFichier.txt");
    try {
        FileOutputStream fos = file.create (FileCreateFlags.PRIVATE);

```

```
        fos.write("Une phrase dans le fichier\n".data);
    } catch (Error e) {
        print("Erreur : %s\n", e.message);
    }
}
```

Compilation :

```
valac --pkg gio-2.0 createFile.vala
```

Exécution :

```
./createFile
```

## 2. Vérifier l'existence d'un fichier

## 3. Suppression d'un fichier

## 2. Interface graphique avec Gtk

Gtk (abréviation de Gimp Toolkit) est une bibliothèque permettant de créer des interfaces graphiques.

### 1. Installation

#### 1 Sous MacOS-X

Pour utiliser la bibliothèque Gtk sous MacOS-X, il faut installer le paquetage Gtk+-3.0 avec Homebrew. Ouvrez un terminal et tapez :

```
brew install gtk+3
```

#### 2 Sous Linux

Sous linux Debian par exemple, tapez la commande suivante :

```
apt-get install libgtk-3-0 libgtk-3-dev
```

#### 3 Sous Windows

Lancez MSYS2 puis dans le terminal tapez :

```
mingw64.exe
```

Un nouveau terminal s'ouvre, tapez la commande suivante dans ce nouveau terminal :

```
pacman -S mingw64/mingw-w64-x86_64-gtk3
```

## 2. Une simple fenêtre

Fichier : main.vala

```
using Gtk;

public class test: Window{
    public test() {
        this.destroy.connect(Gtk.main_quit);
        set_default_size(200,50);
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        var window = new test();
        window.show_all();
        Gtk.main();
        return 0;
    }
}
```

Compilation :

```
valac --pkg gtk+-3.0 main.vala
```

Exécution :

```
./main
```

### 3. Une fenêtre avec un label

L'exemple ci-dessous affiche un label dans une fenêtre.

```
using Gtk;

public class Application: Window{
    public Application() {
        this.destroy.connect(Gtk.main_quit);
        set_default_size(200,50);

        Gtk.Label label = new Gtk.Label("Bonjour le monde!");
        this.add(label);
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        Application app = new Application();
        app.show_all();

        Gtk.main();
        return 0;
    }
}
```

### 4. Une fenêtre avec un label et un Bouton

L'exemple ci-dessous affiche un label et un bouton dans une fenêtre. Lorsque l'on clique sur le bouton, le texte du label change. Créer un fichier « exemple.vala » et copiez le contenu suivant :

```
using Gtk;

public class window: Window{
    private Label label;
    private Button button;

    public window() {
        this.destroy.connect(Gtk.main_quit);
        set_default_size(200,50);

        label = new Gtk.Label("Bonjour le monde!");

        button = new Gtk.Button.with_label("Valider");
        button.clicked.connect( () => {
            label.set_label("Changement de texte!");
        });

        var hbox = new Box(Orientation.VERTICAL, 20);
        hbox.add(label);
        hbox.add(button);
        add(hbox);
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        var win = new window();
        win.show_all();

        Gtk.main();
        return 0;
    }
}
```

Les objets label et button sont placés dans un container (objet hbox → méthode hbox.add). Ce container est ensuite positionné dans la fenêtre de l'application principale ( add(hbox) ).

Pour compiler et exécuter, ouvrez un terminal et taper :

```
valac --pkg gtk+-3.0 exemple.vala && ./exemple
```

## 5. Gestion des évènements

### 1. Événement du clavier

```
/*
 * Compilation:
 * valac eventKey.vala --pkg gtk+-3.0
 */

using Gtk;

public class EventKey: Gtk.Window {
    public EventKey() {
        this.destroy.connect(Gtk.main_quit);
        this.title = "Test Event Key";

        this.key_press_event.connect( (event) => {
            print("Keyval : %u\n", event.keyval);
            print("Hardware Keycode : %u\n", event.hardware_keycode);
            print("Str : %s\n", event.str);
            return true;
        });
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        var evtKey = new EventKey();
        evtKey.show_all();
        Gtk.main();
        return 0;
    }
}
```

## 6. Label, Button et Entry

Cet exemple est constitué d'une fenêtre comportant un label, une boîte d'édition et un bouton. Lorsque nous cliquons sur le bouton, le contenu de la boîte d'édition va s'afficher dans le label.

Fichier : entry.vala

```
/* compilation:
 valac --pkg gtk+-3.0 entry.vala && ./entry
 */

using Gtk;

public class window: Window{

    private Label label;
    private Entry entry;
    private Button button;

    public window() {
        this.destroy.connect(Gtk.main_quit);
        set_default_size(200,50);

        label = new Gtk.Label("Entrez du texte puis cliquez sur Valider");

        entry = new Gtk.Entry();

        button = new Gtk.Button.with_label("Valider");
        button.clicked.connect( () => {
            label.set_label(entry.get_text());
        });

        var hbox = new Box(Orientation.VERTICAL, 20);
        hbox.add(label);
    }
}
```

```

hbox.add(entry);
hbox.add(button);
add(hbox);
}

public static int main(string[] args) {
    Gtk.init(ref args);
    var win = new window();
    win.show_all();

    Gtk.main();
    return 0;
}
}

```

## 7. Les menus

### 2. Position des menus dans la barre principale

Nous créons un `Gtk.MenuBar` (barre principale) qui contiendra les éléments de notre menu (par exemple : Fichier, Édition ...). Nous créons ensuite nos menus et les plaçons dans la barre principale.

menu1.vala

```

using Gtk;

public class TestMenu: Window {
    private Gtk.MenuBar menuBar;
    private Gtk.MenuItem menuItemFile;
    private Gtk.MenuItem menuItemEdit;

    public TestMenu() {
        this.destroy.connect(Gtk.main_quit);
        this.set_default_size(200, 150);

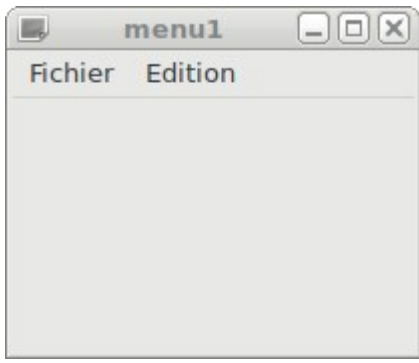
        this.menuBar = new Gtk.MenuBar();
        this.menuItemFile = new Gtk.MenuItem.with_label("Fichier");
        this.menuItemEdit = new Gtk.MenuItem.with_label("Edition");
        this.menuBar.add(this.menuItemFile);
        this.menuBar.add(this.menuItemEdit);

        var mainBox = new Gtk.Box(Gtk.Orientation.VERTICAL, 0);
        mainBox.pack_start(this.menuBar, false, false, 0);
        this.add(mainBox);
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        var win_menu = new TestMenu();
        win_menu.show_all();
        Gtk.main();
        return 0;
    }
}

```

Capture d'écran :





## 8. Le widget liste ou tableau – TreeView

Le widget liste fourni par Gtk est extrêmement élaboré. On peut insérer n'importe quel type d'élément dans un **TreeView**, des nombres, du texte mais aussi des objets (case à cocher et boutons par exemple). Cette puissance engendre néanmoins une certaine complexité.

Pour créer une liste, même élémentaire, nous avons au minimum, besoin de trois objets Gtk : Gtk.TreeIter, Gtk.ListStore et Gtk.TreeView.

**TreeIter** correspond à l'élément de base qui permettra de communiquer avec notre modèle de liste.

**ListStore** contiendra l'objet **TreeIter** avec les différents éléments de la liste. Par exemple pour une liste contenant un seul élément de type string nous déclarerons l'objet ListStore de cette manière :

```
var list_store = new Gtk.ListStore(1, typeof(string)) ;
```

Une liste peut comporter plusieurs colonnes. Ce qui permet de générer des tableaux. Si la liste contient deux colonnes dont le premier est un entier et le deuxième un type string, on écrira alors :

```
var list_store = new Gtk.ListStore(2, typeof(int), typeof(string)) ;
```

et ainsi de suite.

On ajoute ensuite un élément TreeIter dans un ListStore en utilisant la méthode **append** du ListStore. Nous définissons le type d'élément et sa position dans la liste en utilisant la méthode **set**. Par exemple :

```
list_store.set(iter, 0, "toto");
```

signifie que nous plaçons le texte « toto » dans la première colonne (on notera que la première colonne commence par 0 et pas par 1).

Nous plaçons ensuite l'objet ListStore dans le TreeView.

Un exemple simple est illustré ci-dessous.

### 3. Une liste simple comportant un seul élément

Fichier : treeView.vala

```
using Gtk;

public class viewWindow:Window {

    public viewWindow() {
        this.destroy.connect(Gtk.main_quit);
        set_default_size(200,150);

        Gtk.ListStore list_store=new Gtk.ListStore(1, typeof(string));
        Gtk.TreeIter iter;

        list_store.append(out iter);
        list_store.set(iter, 0, "toto");
        list_store.append(out iter);
        list_store.set(iter, 0, "bibi");

        Gtk.TreeView view=new TreeView.with_model(list_store);
        this.add(view);

        Gtk.CellRendererText cell=new Gtk.CellRendererText();
        view.insert_column_with_attributes(-1,"Nom", cell, "text", 0);
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        var view = new viewWindow();
        view.show_all();
        Gtk.main();
        return 0;
    }
}
```

### 4. Afficher les éléments d'un TreeView

On reprend le même exemple que précédemment mais cette fois-ci, on rajoute un label qui va afficher les valeurs de notre TreeView. Pour cela on utilise la méthode **@foreach** de l'objet **TreeModel**. Cette méthode renvoie la valeur **false** tant qu'il reste des éléments à parcourir.

```
using Gtk;

public class viewWindow:Window {
```

```

public viewWindow() {
    this.destroy.connect(Gtk.main_quit);
    set_default_size(200,150);

    Gtk.ListStore list_store=new Gtk.ListStore(1, typeof(string));
    Gtk.TreeIter iter;

    list_store.append(out iter);
    list_store.set(iter, 0, "toto");
    list_store.append(out iter);
    list_store.set(iter, 0, "bibi");
    list_store.append(out iter);
    list_store.set(iter, 0, "fafa");

    Gtk.TreeView view=new TreeView.with_model(list_store);

    Gtk.CellRendererText cell=new Gtk.CellRendererText();
    view.insert_column_with_attributes(-1,"Nom", cell, "text", 0);

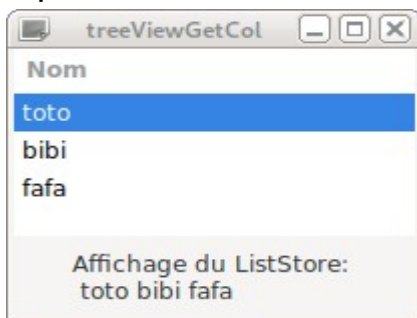
    string text_label="Affichage du ListStore: \n";
    var model = view.get_model();
    model.@foreach( (model, path, iter) => {
        Value val;
        model.get_value(iter, 0, out val);
        text_label = text_label + " " + val.get_string();
        return false;
    });
    Gtk.Label labelInfo = new Gtk.Label(text_label);

    Gtk.Box box = new Gtk.Box(Gtk.Orientation.VERTICAL, 0);
    box.pack_start(view);
    box.pack_start(labelInfo);
    this.add(box);
}

public static int main(string[] args) {
    Gtk.init(ref args);
    var view = new viewWindow();
    view.show_all();
    Gtk.main();
    return 0;
}
}

```

Capture d'écran :



## 5. Accéder aux éléments d'un TreeView lors d'un clique de souris

Pour accéder aux éléments d'un **TreeView**, lors d'une sélection avec la souris, nous devons utiliser les objets **model** et **selection**. Il n'est pas nécessaire de créer ces objets avec la directive **new** car ceux-ci sont automatiquement engendrés lors de la création d'un **TreeView**. Nous utiliserons les méthodes **get\_model()** et **get\_selection()** de l'objet **TreeView**.

L'exemple ci-dessous génère un TreeView contenant des noms et des prénoms. Lorsque l'on clique sur une ligne, les caractéristiques de la ligne s'affiche dans des labels. Le numéro de ligne est obtenu en utilisant la méthode `get_string_from_iter()` de l'objet `model`.

Fichier: listStore.vala

```
using Gtk;

public class viewWindow:Window {
    private Gtk.Label label_index;
    private Gtk.Label label_value;

    public viewWindow() {
        this.destroy.connect(Gtk.main_quit);
        set_default_size(200,150);

        Gtk.ListStore list_store=new Gtk.ListStore(2, typeof(string), typeof(string));
        Gtk.TreeIter iter;

        list_store.append(out iter);
        list_store.set(iter, 0, "Nouvel",1, "Alexandre");
        list_store.append(out iter);
        list_store.set(iter, 0, "Limouzy", 1, "Agathe");
        list_store.append(out iter);
        list_store.set(iter, 0, "Kuntz", 1, "Kilian");
        list_store.append(out iter);
        list_store.set(iter, 0, "Thi Quynh", 1, "Trang");

        Gtk.TreeView view=new TreeView.with_model(list_store);

        Gtk.CellRendererText cell=new Gtk.CellRendererText();
        view.insert_column_with_attributes(-1,"Nom", cell, "text", 0);
        view.insert_column_with_attributes(-1, "Prénom", cell, "text", 1);
        view.set_activate_on_single_click(true);
        view.row_activated.connect( () => {
            var model = view.get_model();
            var selection = view.get_selection();
            selection.get_selected(out model, out iter);
            var str_index = model.get_string_from_iter(iter);
            Value name, surname;
            model.get_value(iter, 0, out name);
            model.get_value(iter, 1, out surname);
            label_index.set_text("Index="+str_index);
            label_value.set_text(
                "Nom: " + name.get_string() + "\n"+
                "Prénom: " + surname.get_string());
        });

        label_index = new Gtk.Label("Information index");
        label_value = new Gtk.Label("Information valeur");

        var boxLabel = new Gtk.Box(Gtk.Orientation.VERTICAL, 10);
        boxLabel.pack_start(label_index);
        boxLabel.pack_start(label_value);

        var box = new Gtk.Box(Gtk.Orientation.HORIZONTAL, 10);
        box.pack_start(view);
        box.pack_start(boxLabel);
        this.add(box);
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        var view = new viewWindow();
        view.show_all();
        Gtk.main();
        return 0;
    }
}
```

## Compilation

```
valac listStore.vala -pkg gtk+-3.0
```

## Capture d'écran



## 6. Éditer une cellule d'un TreeView

Nous modifions le champ d'une cellule en utilisant la méthode **set()** de l'objet **ListStore**.

Fichier : TreeViewEdit.vala

```
using Gtk;

public class viewWindow:Window {
    public viewWindow() {
        this.destroy.connect(Gtk.main_quit);
        set_default_size(200,150);

        Gtk.ListStore list_store=new Gtk.ListStore(1, typeof(string));
        Gtk.TreeIter iter;

        list_store.append(out iter);
        list_store.set(iter, 0, "Earth");
        list_store.append(out iter);
        list_store.set(iter, 0, "Mars");

        Gtk.TreeView view=new TreeView.with_model(list_store);
        this.add(view);

        Gtk.CellRendererText cell=new Gtk.CellRendererText();
        view.insert_column_with_attributes(-1, "Planet", cell, "text", 0);
        cell.editable = true;

        cell.edited.connect ((path, data) => {
            Gtk.TreePath tPath = new Gtk.TreePath.from_string(path);
            var model = view.get_model();
            var res = model.get_iter(out iter, tPath);
            if (res == true) {
                list_store.set(iter, 0, data);
            }
        });
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        var view = new viewWindow();
        view.show_all();
        Gtk.main();
        return 0;
    }
}
```

Compilation et exécution :

```
valac treeViewEdit.vala --pkg gtk+-3.0 && ./treeViewEdit
```

## 7. Éditer une cellule d'un treview comportant plusieurs colonnes

```
using Gtk;

public class viewWindow:Window {
    private Gtk.TreeView view;
    private Gtk.ListStore list_store;
    private Gtk.TreeIter iter;

    private void cell_edited(Gtk.CellRendererText cell, string path, string data, int column) {
        Gtk.TreePath tPath = new Gtk.TreePath.from_string(path);
        var model = view.get_model();
        var res = model.get_iter(out iter, tPath);
        if (res == true) {
            list_store.set(iter, column, data);
        }
    }

    public viewWindow() {
        this.destroy.connect(Gtk.main_quit);
        set_default_size(200,150);

        this.list_store=new Gtk.ListStore(2, typeof(string), typeof(string));
        list_store.append(out iter);
        list_store.set(iter, 0, "Earth", 1, "6400");
        list_store.append(out iter);
        list_store.set(iter, 0, "Mars", 1, "3400");

        this.view=new TreeView.with_model(list_store);
        this.add(view);

        Gtk.CellRendererText cell;

        cell=new Gtk.CellRendererText();
        cell.editable = true;
        view.insert_column_with_attributes(-1, "Planet", cell, "text", 0);
        cell.edited.connect ((path, data) => {cell_edited(cell, path, data, 0)});

        cell=new Gtk.CellRendererText();
        cell.editable = true;
        view.insert_column_with_attributes(-1, "Ray", cell, "text", 1);
        cell.edited.connect ((path, data) => {cell_edited(cell, path, data, 1)});
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        var view = new viewWindow();
        view.show_all();
        Gtk.main();
        return 0;
    }
}
```

**Capture d'écran**

Planet	Ray
Earth	6400
Jupiter	69900

## 9. Le widget ListBox

Ce widget est une simplification du widget Treeview. On peut insérer n'importe quel objet dans un ListBox. Si on veut insérer du texte on devra utiliser un objet **label**.

### 1 Insérer des éléments

L'exemple suivant affiche une liste de trois animaux. Lorsque l'on clique sur un animal, on affiche la valeur dans le terminal.

**ListBox.vala :**

```
using Gtk;

public class Application: Window{
    private Gtk.ListBox listbox;

    public Application() {
        this.destroy.connect(Gtk.main_quit);
        set_default_size(200,50);

        string[] atext={
            "Oiseau",
            "Chien",
            "Hamster"
        };
        this.listbox = new Gtk.ListBox();
        var label1 = new Gtk.Label(atext[0]);
        var label2 = new Gtk.Label(atext[1]);
        var label3 = new Gtk.Label(atext[2]);
        listbox.insert(label1, 0);
        listbox.insert(label2, 1);
        listbox.insert(label3, 2);

        this.listbox.row_activated.connect( () => {
            print("Clique\n");
            Gtk.ListBoxRow lbr = this.listbox.get_selected_row();
            int index = lbr.get_index();
            print("Select = %d\t Text=%s\n", index, atext[index]);
        });

        this.add(listbox);
    }

    public static int main(string[] args) {
        Gtk.init(ref args);
        Application app = new Application();

        app.show_all();

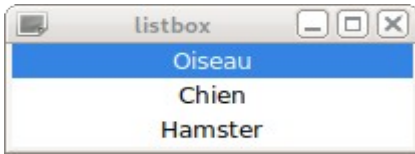
        Gtk.main();
        return 0;
    }
}
```

```
}  
}
```

### Compilation et exécution :

```
valac Listbox.vala && ./Listbox
```

### Capture d'écran :



## 2 Effacement et ajout d'éléments dans une liste

### ListboxAddClear.vala :

```
using Gtk;  
  
public class Application: Window{  
    private Gtk.ListBox listbox;  
    private Gtk.Entry entry;  
    private Gtk.Button buttonAdd;  
    private Gtk.Button buttonClear;  
  
    private string text;  
  
    public Application() {  
        this.destroy.connect(Gtk.main_quit);  
        set_default_size(200,50);  
  
        string[] atext={  
            "Mercure",  
            "Venus",  
            "Terre"  
        };  
        this.listbox = new Gtk.ListBox();  
        var label1 = new Gtk.Label(atext[0]);  
        var label2 = new Gtk.Label(atext[1]);  
        var label3 = new Gtk.Label(atext[2]);  
        listbox.insert(label1, 0);  
        listbox.insert(label2, 1);  
        listbox.insert(label3, 2);  
  
        this.listbox.row_activated.connect( () => {  
            print("Clique\n");  
            Gtk.ListBoxRow lbr = this.listbox.get_selected_row();  
            int index = lbr.get_index();  
            print("Select = %d\t Text=%s\n", index, atext[index]);  
        });  
  
        this.entry = new Gtk.Entry();  
  
        this.buttonAdd = new Gtk.Button.with_label("Ajouter");  
  
        this.buttonClear = new Gtk.Button.with_label("Effacer");  
  
        var box = new Gtk.Box(Gtk.Orientation.VERTICAL, 0);  
        box.pack_start(this.listbox, true, true, 0);  
        box.pack_start(this.entry, false, false, 0);  
        box.pack_start(this.buttonAdd, false, false, 0);  
        box.pack_start(this.buttonClear, false, false, 0);
```

```

        this.add(box);

//signal connect
this.buttonAdd.clicked.connect( () => {
    print("Ajout de nouveaux elements\n");
    this.text = this.entry.get_text();
    print("Text : %s\n", this.text);
    var labelE = new Gtk.Label(this.text);
    listBox.insert(labelE, -1);
    this.show_all();
});

this.buttonClear.clicked.connect( () => {
    print("Effacement de la liste\n");
    foreach (Gtk.Widget element in this.listBox.get_children()) {
        this.listBox.remove(element);
    }
});
}

public static int main(string[] args) {
    Gtk.init(ref args);
    Application app = new Application();
    app.show_all();
    Gtk.main();
    return 0;
}
}

```

## 10. Le widget ComboBoxText

1

obtenir le texte sélectionné

```

/*
 * Compilation & execution
 * valac --pkg gtk+-3.0 test_combo.vala && ./test_combo
 */

using Gtk;

public class TestComboBox : Gtk.Window {
    private Gtk.Label label;
    private Gtk.ComboBoxText comboBox;

    public TestComboBox() {
        this.title = "Test ComboBox";
        this.destroy.connect(Gtk.main_quit);

        this.label = new Gtk.Label("Sélectionner un oiseau");
        this.comboBox = new Gtk.ComboBoxText();
        this.comboBox.append_text("pigeon");
        this.comboBox.append_text("moineau");
        this.comboBox.append_text("corbeau");
        this.comboBox.active = 0;

        this.comboBox.changed.connect( ()=> {
            print("Changement %s\n", this.comboBox.get_active_text());
            this.label.set_text("Vous avez sélectionné le " +
                this.comboBox.get_active_text());
        });

        var box = new Gtk.Box(Orientation.VERTICAL, 10);
        box.pack_start(this.label);
        box.pack_start(this.comboBox);
        this.add(box);
    }
}

```

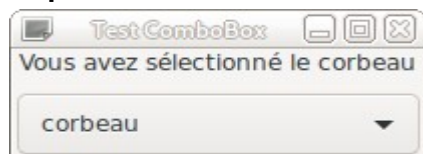


```

    public static int main(string[] args) {
        Gtk.init(ref args);
        TestComboBox tcb = new TestComboBox();
        tcb.show_all();
        Gtk.main();
        return 0;
    }
}

```

Capture d'écran :



## 11. Le widget Notebook

Fichier : notebook.vala

```

using Gtk;

public class Application: Window {
    public Application() {
        this.destroy.connect(Gtk.main_quit);

        var labelInfo = new Gtk.Label("Info ...");

        var labelPage1 = new Gtk.Label("Pluie");
        var labelPage2 = new Gtk.Label("Soleil");
        var labelPage3 = new Gtk.Label("Bof");

        var label1 = new Gtk.Label("\nAujourd'hui il fait\npas beau!!\n");
        var label2 = new Gtk.Label("\nAujourd'hui il\nfait très beau!\n");
        var label3 = new Gtk.Label("\nC'est mitigé aujourd'hui!\n");

        var notebook = new Gtk.Notebook();
        notebook.append_page(label1, labelPage1);
        notebook.append_page(label2, labelPage2);
        notebook.append_page(label3, labelPage3);

        var box = new Gtk.Box(Gtk.Orientation.VERTICAL, 0);
        box.pack_start(notebook, false, false, 0);
        box.pack_start(labelInfo, false, false, 0);
        this.add(box);

        notebook.switch_page.connect( (page, page_num) => {
            labelInfo.set_text(
                "Numéro d'onglet : " + page_num.to_string() + "\n"+
                "Texte de l'onglet : " + notebook.get_tab_label_text(page)
            );
        });
    }
}

public static int main(string[] args) {
    Gtk.init(ref args);

    Application app = new Application();
    app.show_all();

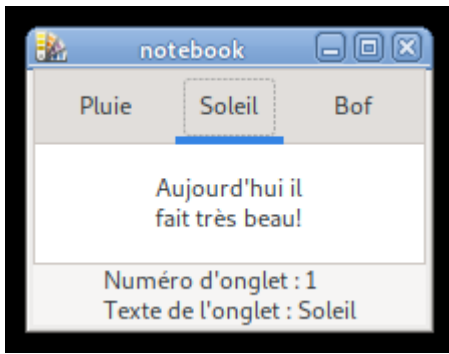
    Gtk.main();
    return 0;
}

```

Compilation et exécution

```
valac --pkg gtk+-3.0 notebook.vala
```

Capture d'écran



## 12. Le widget Calendar

### Fichier calendar.vala

```
using Gtk;

public class TestCalendar: Window {
    private uint year;
    private uint month;
    private uint day;

    public TestCalendar() {
        this.destroy.connect(Gtk.main_quit);

        var cal = new Gtk.Calendar();
        var label = new Gtk.Label("Date format : jour/mois/ans");

        var box = new Gtk.Box(Gtk.Orientation.VERTICAL, 0);
        box.pack_start(cal, false, false, 0);
        box.pack_start(label, false, false, 0);

        this.add(box);

        cal.day_selected.connect( () => {
            cal.get_date(out this.year, out this.month, out this.day);
            month += 1;
            string text_date = "Le " + day.to_string() + "/" + month.to_string()+"/"+year.to_string();
            label.set_text(text_date);
        });
    } //end TestCalendar
} //end class

public static int main(string[] args) {
    Gtk.init(ref args);

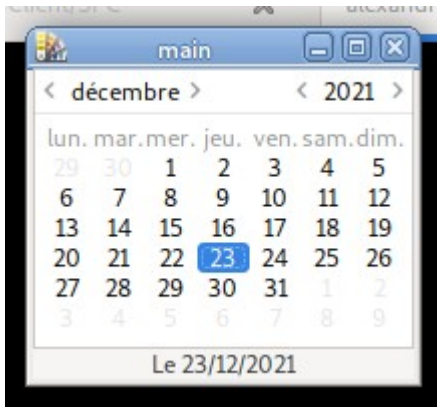
    TestCalendar tc = new TestCalendar();
    tc.show_all();

    Gtk.main();
    return 0;
}
```

### Compilation et exécution

```
valac --pkg gtk+-3.0 calendar.vala && ./calendar
```

### Capture d'écran



## 3. Graphismes et manipulation des différents formats avec Cairo

### 1. Installation de Cairo

Sous Windows :

```
pacman -S mingw-w64-x86_64-cairo
```

### 2. Générations d'un document pdf

Nous allons voir dans cet exemple comment générer un document pdf au format A4. Pour une résolution de 200DPI, le format A4 correspond à une taille de 1654x2339 pixels.

Fichier : creer\_pdf.vala

```
using Cairo;

public class CreerPdf {

    static int main(string[] args) {
        string filename="test.pdf";
        Cairo.Surface surface = new PdfSurface (filename, 1654, 2339);
        Cairo.Context context = new Cairo.Context(surface);
        context.move_to(30,30);
        context.show_text("Bonjour, ceci est un test!");
        context.show_page();
        return 0;
    }
}
```

Compilation

```
valac --pkg cairo creer_pdf.vala
```

Exécution

```
./creer_pdf
```

## 4. Base de donnée locale avec Sqlite

### 1. Introduction

**Sqlite** est une bibliothèque permettant de gérer une base de donnée locale en s'appuyant sur la norme du langage Sql. Pour visualiser et créer une base de donnée avec un logiciel dédié, vous pouvez utiliser « **DB Browser For Sqlite** »

### 2. Installation de Sqlite

Sous Windows :

```
pacman -S mingw-w64-x86_64-sqlite3
```

### 3. Remarques sur Sqlite

Sqlite a une particularité à respecter lorsque l'on crée une table. Il s'occupe automatiquement de la création de la clé primaire. Ce n'est donc pas la peine le spécifier dans la requête de création.

Par exemple on se contentera d'écrire :

```
CREATE TABLE Customer (name TEXT, surname TEXT)
```

sans préciser l'existence d'une clé primaire.

## 4. Création d'une base de donnée

L'exemple suivant crée un fichier de base de donnée nommé **name\_database.db**.

Fichier : createDatabase.vala

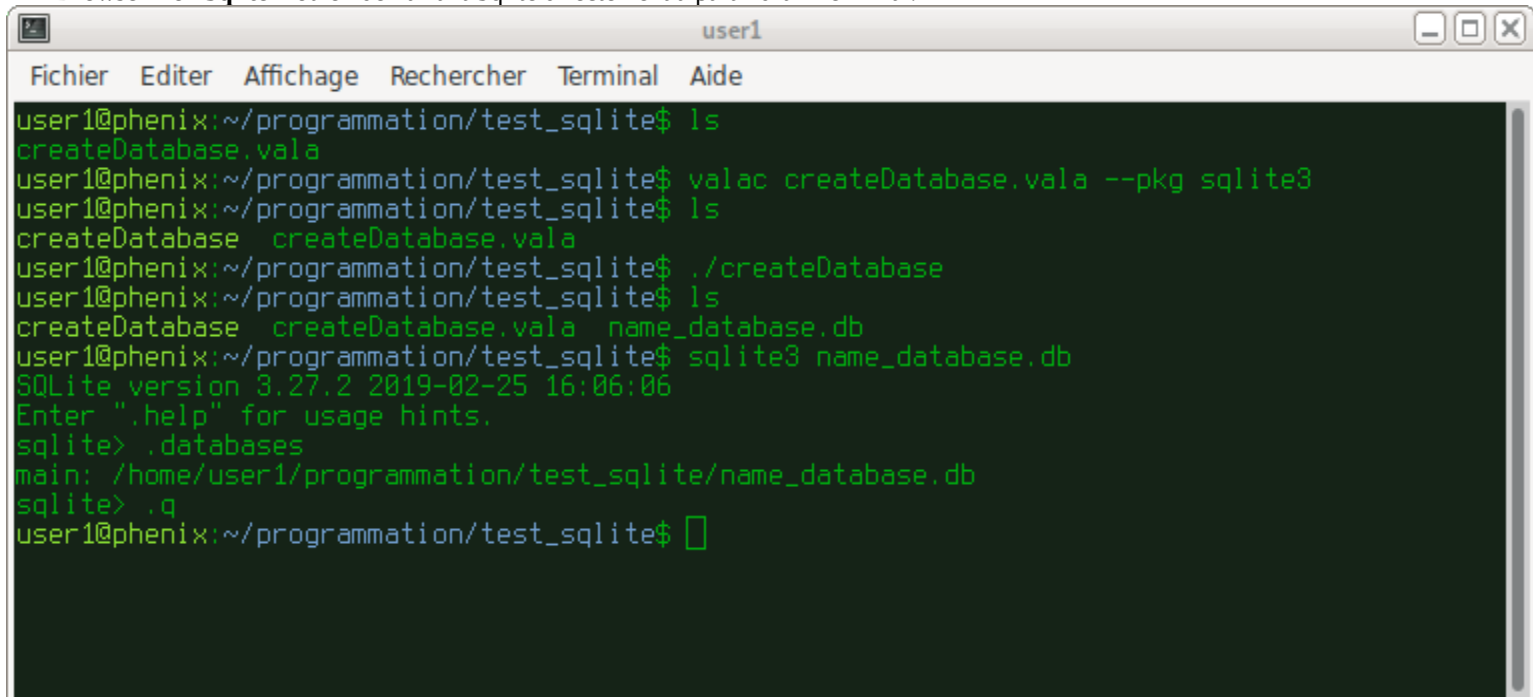
```
/*
 * Compilation :
 * valac createDatabase.vala --pkg sqlite3
 */

using Sqlite;

int main() {
    Sqlite.Database db;

    int result = Sqlite.Database.open ("name_database.db", out db);
    if (result != Sqlite.OK) {
        stderr.printf("Erreur : %d: %s\n", db.errcode(), db.errmsg());
        return -1;
    }
    return 0;
}
```

On peut vérifier que le fichier a bien été créé et qu'il est compatible avec sqlite3 soit en utilisant un logiciel dédié comme « **DB Browser For SQLite** » ou en démarrant SQLite directement à partir d'un Terminal.



The screenshot shows a terminal window titled 'user1' with a menu bar containing 'Fichier', 'Editer', 'Affichage', 'Rechercher', 'Terminal', and 'Aide'. The terminal output is as follows:

```
user1@phenix:~/programmation/test_sqlite$ ls
createDatabase.vala
user1@phenix:~/programmation/test_sqlite$ valac createDatabase.vala --pkg sqlite3
user1@phenix:~/programmation/test_sqlite$ ls
createDatabase  createDatabase.vala
user1@phenix:~/programmation/test_sqlite$ ./createDatabase
user1@phenix:~/programmation/test_sqlite$ ls
createDatabase  createDatabase.vala  name_database.db
user1@phenix:~/programmation/test_sqlite$ sqlite3 name_database.db
SQLite version 3.27.2 2019-02-25 16:06:06
Enter ".help" for usage hints.
sqlite> .databases
main: /home/user1/programmation/test_sqlite/name_database.db
sqlite> .q
user1@phenix:~/programmation/test_sqlite$
```

## 5. Création d'une table

## 6. Afficher des données

## 5. Rendu html avec WebKit

L'exemple ci-dessous affiche une fenêtre avec du contenu html.

```
using Gtk;
using WebKit;

public class TestWebKit: Window {
    private WebView web_view;

    public TestWebKit() {
        this.title = "Test WebKit";
    }
}
```

```

    this.set_default_size(500, 200);
    this.destroy.connect(Gtk.main_quit);

    this.web_view = new WebView();
    this.web_view.load_html("<h1>Voici un titre</h1>
                            <p>Et ceci est un paragraphe</p>", null);
    this.add(web_view);
}

public static int main(string[] args) {
    Gtk.init(ref args);
    var twb = new TestWebKit();
    twb.show_all();
    Gtk.main();
    return 0;
}
}

```

#### Compilation :

En supposant que vous avez écrit le code ci-dessus dans un fichier nommé main.vala, la compilation donne :

```
valac --pkg gtk+-3.0 --pkg webkit2gtk-4.0 main.vala
```

#### Capture d'écran :



## 12. Faire appel à des fonctions du langage C

On peut directement faire appel à des fonctions du langage C à l'intérieur du code Vala. Si on passe des arguments à l'intérieur des fonctions, il faudra faire attention d'employer (dans le fichier Vala) des variables correspondantes aux équivalents du langage C (par exemple un **string** pour un **char \***). Dans le cas où les arguments sont regroupés dans des structures du langage C, il faudra également faire quelques adaptations (par exemple, il n'y a pas de pointeurs en Vala). Lorsque nous faisons appel à des fonctions C dans le code source Vala, il faudra faire précéder les fonctions C par le mot **extern**.

Prenons les exemples suivants :

### 1.Code C

Fichier : prog.c

```

#include <stdio.h>

typedef struct {
    int nb;
    char *str;
}Data;

void display() {
    printf("Hello C!\n");
}

void display_number(int nb) {
    printf("Nombre : %i\n", nb);
}

void display_data(Data *dat) {
    printf("Data nb=%i\n", dat->nb);
    printf("Data str=%s\n", dat->str);
}

```

## 2.Code Vala

Fichier : mix.vala

```

extern struct Data {
    int nb;
    string str;
}

extern void display();
extern void display_number(int nb);
extern void display_data(Data data);

int main() {
    int i=43;
    Data data=Data();
    data.nb=56;
    data.str="Cool!";
    display();
    display_number(i);
    display_data(data);
    return 0;
}

```

## 3.Compilation

```
valac mix.vala prog.c
```

## 4.Exécution

```
./mix
```

## 5.Résultat

```

Hello C!
Nombre : 43
Data nb=56
Data str=Cool!

```

# 13. Empaquetage de logiciel crée avec Vala

## 1.Empaquetage sous Windows

Si vous essayer d'exécuter le binaire en dehors de l'environnement de MSYS2 vous devriez avoir des messages d'erreurs de dll manquants. Ce sont des dll propre à l'environnement Vala qui sont pour la plupart situé dans des répertoires lib de mingw.

Pour récupérer les dll manquants sous Windows :

Lancez msys2. Un terminal s'ouvre et tapez la commande suivante :

```
mingw64.exe
```

Un deuxième terminal s'ouvre. Tapez la commande suivante dans le deuxième terminal :

```
ldd executable.exe | grep mingw | awk '{print $3}' > dll.txt
```

Un fichier **dll.txt** a été créé contenant les chemins de tous les dll manquants. Créer un répertoire **dir\_dll** en tapant :

```
mkdir dir_dll
```

On va copier les dll manquants dans ce répertoire :

```
for i in $(cat dll.txt); do echo $i; cp $i dir_dll;done
```

Si vous copiez votre exécutable dans ce répertoire et que vous le lancez à partir de ce répertoire, vous ne devriez plus avoir de message d'erreur.

## 14. Annexe

### 1. Convention de nommage en Vala

Classes, structs, delegate types : CamelCase

méthodes, propriétés, signals : lower\_case

variables locales, champs (fields) : lower\_case

constantes, type énumérés : UPPER\_CASE

### 2. Commandes shell élémentaires

Cette partie ne traitera pas en détail l'utilisation du shell. Il ne donnera que le minimum nécessaire, afin que le lecteur puisse mettre en pratique les exemples de ce tutoriel. Les noms des commandes du shell font appelent à des d'abréviations (par exemple la commande **ls** est l'abréviation de **list**). Cela pourra vous aider à mieux retenir le nom des commandes.

#### 1. Savoir dans quel répertoire on se situe - **pwd**

La commande **pwd** permet de connaître le répertoire courant dans lequel on se situe.

#### 2. Afficher le contenu d'un répertoire - **ls**

La commande **ls** permet de lister le contenu du répertoire courant. Pour connaître toutes les options de **ls**, taper **man ls** dans un terminal.

#### 3. Effacer l'écran - **clear** et **reset**

La commande **clear** efface l'écran du terminal. On peut toutefois revenir à l'historique des commandes précédentes en utilisant la barre de défilement du terminal. La commande **reset** efface non seulement l'écran mais aussi toute les commandes précédentes.

#### 4. Ce déplacer dans un répertoire - **cd**

La commande **cd** (abréviation de change directory) permet de changer de répertoire.

##### 1 Chemin absolu

Le chemin absolu part toujours de la racine (symbole **/**). Dans l'exemple suivant on se déplace dans le répertoire « programmation » de l'utilisateur « user1 » en partant directement de la racine :

```
cd /home/user1/programmation
```

Cette commande fonctionnera quelque soit l'endroit où on se trouve car on part de la racine. Elle nous mènera obligatoirement dans le répertoire programmation (à condition d'avoir les bons droits).

##### 2 Chemin relatif

Le chemin relatif part du répertoire courant. Par exemple la commande suivante :

```
cd programmation
```

Ne fonctionnera que si l'on se trouve dans le répertoire **/home/user1**. Si on se trouve dans un répertoire au dessus alors elle ne fonctionnera pas. La commande **ls** affichera les répertoires qui sont accessibles en utilisant un chemin relatif.

##### 3 Monter d'un répertoire

On monte dans le répertoire du dessus avec la commande :

```
cd ..
```

(notez qu'il faut un espace entre **cd** et le premier point)

#### 5. Créer un répertoire - **mkdir**

La commande **mkdir** (abréviation de make directory) crée un répertoire.

```
mkdir toto
```

crée un répertoire nommé **toto**.

On peut créer plusieurs répertoires d'affilés :

```
mkdir toto titi
```

crée deux répertoires nommés **toto** et **titi**.



## 6. Supprimer un répertoire – rmdir

La commande **rmdir** supprime un répertoire. Pour supprimer un répertoire avec cette commande il faut que le répertoire soit vide. Si vous voulez supprimer un répertoire et tout son contenu utilisez la commande **rm** avec l'option **-R** (récuratif).

## 3. Commandes Sqlite élémentaires

### 1. Démarrer Sqlite

Dans un terminal taper :

```
sqlite3 nom_database.db
```

### 2. Visualiser les bases de données

```
.databases
```

### 3. Afficher les tables

```
.tables
```

### 4. Visualiser l'architecture d'une table

```
.schema
```

### 5. Quitter Sqlite

```
.q
```

## Index lexical

Aléatoire (générer un nombre).....	15
Générations d'un document pdf.....	39
Suppression de Widget.....	<b>38</b>