DU 11 AU 18 MARS 2025 | N°43

# CLOUDY MAG

**ENVIE D'ETRE LE DEV MOBILE QUE TU PENSES ÊTRE ?**
Le Guide de Cloudy pour s'en sortir.

**ICI C'EST PARIS !**
Ce soir Victoire obligatoire pour le meilleur club de France

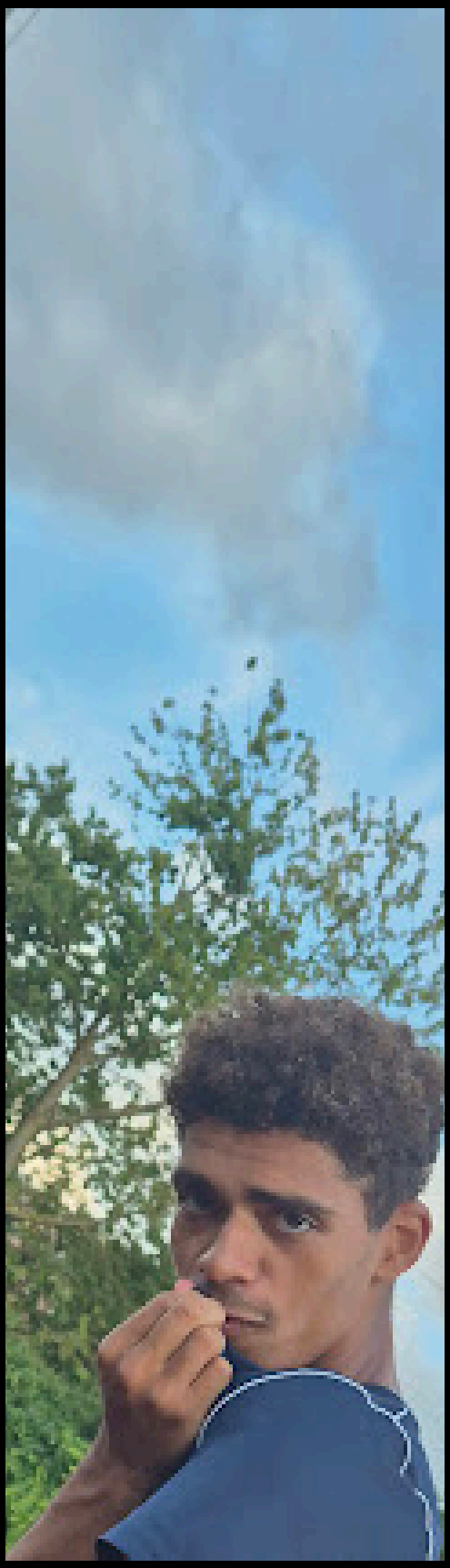
## BASH.

OU COMMENT PROTEGER SA VOITURE OU SON SYSTEME
(BASH = BACHE HAHAHAHA)

# SOMMAIRE

**N °43**

# EDITO



N°43 DU 11 AU 18 MARS 2025

Ca faisait un petit moment non ?
On reprend les bonnes habitudes, un petit Cloudy sur les deux ctp qui nous attendent cette semaine. Deux beaux morceaux, mais c'est abordable.

P.S Le meilleur club de france joue ce soir, evidemment le PSG va gagner sinon, je m'appelle pas Cloudy. But de Barcola et Kvara retenez bien.

Patrick Cloudy
Président de Cloudy Mag

# BASH


## Partie 1 : Bases du Bash (TP1)

### 1.1. Manipulation des fichiers et répertoires

Cette section couvre les commandes de base pour manipuler des fichiers et répertoires.

**Commandes essentielles :**

`cat` : Affiche ou crée un fichier.
`sort` : Trie les lignes d'un fichier (alphabétiquement ou numériquement avec `-n`).
`nano` : Éditeur de texte simple en ligne de commande.
`mkdir` : Crée un répertoire.
`chmod` : Modifie les permissions d'un fichier ou répertoire (ex. `a-r` retire le droit de lecture à tous).
`ls` : Liste les fichiers et répertoires (avec `-l` pour les détails, `-d` pour les répertoires eux-mêmes).
`du` : Calcule l'espace disque utilisé (ex. `-a` pour tous les fichiers, `-s` pour un sommaire).

**Explications et exemples :**

**Créer et trier un fichier :**

```
cat > test-sort << EOF
211
9
53
97
1000
EOF
sort test-sort        # Trie alphabétiquement
sort -n test-sort     # Trie numériquement : 9 53 97 211 1000
```

Ici, `cat` avec `<< EOF` crée un fichier multiligne, et `sort` organise son contenu.

**Modifier les permissions :**

```
mkdir ~/test-chmod
chmod a-r ~/test-chmod    # Retire la lecture à tous
ls -ld ~/test-chmod       # Affiche : drwx--x--x (plus de 'r' pour autres)
```

`chmod a-r` affecte tous les utilisateurs (`a` = all).

**Gérer les erreurs :**

```
ls ~/test-chmod 2> /dev/null  # Les erreurs sont ignorées
```

`2>` redirige la sortie d'erreur vers `/dev/null` (poubelle virtuelle).

**Analyser l'espace disque :**

```
bash
du -a ~ | sort -n | head -n 10  # Affiche les 10 plus petits fichiers/répertoires
```

`du -a` liste tout, `sort -n` trie par taille, `head -n 10` limite à 10 lignes.

**Redirections (sorties et entrées) :**
Les redirections permettent de contrôler où vont les sorties ou d'où viennent les entrées.

`>` : Redirige la sortie standard vers un fichier (écrase le contenu existant).

```
echo "Bonjour" > salut.txt  # Crée ou écrase salut.txt avec "Bonjour"
```

`>>` : Redirige la sortie standard vers un fichier (ajoute à la fin sans écraser).

```
echo "Salut" >> salut.txt   # Ajoute "Salut" après "Bonjour"
```

`<` : Redirige l'entrée standard depuis un fichier.

```
sort < test-sort  # Trie le contenu de test-sort et l'affiche
```

`<<` : Utilise un "here document" pour fournir une entrée multiligne jusqu'à un délimiteur (ex. `EOF`).

```
cat << EOF > multi.txt
Ligne 1
Ligne 2
EOF  # Crée multi.txt avec deux lignes
```

**1.2. Variables et environnements**
Les variables permettent de stocker et manipuler des données.

**Définition et utilisation :**
Une variable est définie avec `nom=valeur` (sans espaces autour de `=`).
On l'affiche avec `$nom` ou `${nom}`.
**Exemples :**
**Variables simples :**

```
VAR="*Un premier test*"
echo "$VAR"               # Affiche : *Un premier test*
VAR="$VAR*Un deuxième test*"
echo "$VAR"               # Affiche : *Un premier test**Un deuxième test*
```

Les guillemets autour de `$VAR` évitent des problèmes avec les espaces ou caractères spéciaux.
**Variables d'environnement :**

```
printenv SHELL  # Ex. : /bin/bash (shell actuel)
printenv PWD    # Répertoire courant (ex. : /home/user)
printenv OLDPWD # Répertoire précédent
printenv PATH   # Chemins des exécutables (ex. : /usr/bin:/bin)
```

**Substitution avec valeurs par défaut :**
`${A:-valeur}` : Utilise `valeur` si `A` n'est pas définie, sans modifier `A`.
`${A:=valeur}` : Utilise `valeur` et affecte `valeur` à `A` si `A` n'est pas définie.

```
A=abc
echo ${A:-123}  # Affiche : abc
unset A         # Supprime A
echo ${A:-123}  # Affiche : 123 (A reste non défini)
echo ${A:=123}  # Affiche : 123 et A devient 123
echo $A         # Affiche : 123
```

# Partie 2 : Scripting Bash (TP2)

### 2.1. Syntaxe de base des scripts

Un script Bash est un fichier texte contenant des commandes exécutables.

**Structure d'un script :**

Commence par `#!/bin/bash` (shebang, indique l'interpréteur).
Arguments : `$1`, `$2`, … (premier, deuxième argument).
`$#` : Nombre d'arguments.
`$*` ou `$@` : Tous les arguments.
`shift [n]` : Décale les arguments (ex. `$2` devient `$1`).
`exit n` : Termine avec un code de retour `n`.

**Exemple de script (`exemple.sh`) :**

```
#!/bin/bashecho
$1 $2 $3  # Affiche les 3 premiers arguments
shift          # Décale : $2 devient $1, etc.
echo "$*"      # Affiche tous les arguments restants
echo $#        # Nombre d'arguments restants
exit 1         # Termine avec échec
```

Rendre exécutable : `chmod u+x exemple.sh`
Exécuter : `./exemple.sh arg1 arg2 arg3`

**Méthodes d'exécution :**

`./script.sh` : Dans un sous-shell (n'affecte pas le shell courant).
`source ./script.sh` ou `. ./script.sh` : Dans le shell courant (modifications persistantes).

### 2.2. Portée des variables

La portée détermine où une variable est accessible.

**Variables locales vs exportées :**

Sans `export` : Locale au shell courant.
Avec `export` : Disponible dans les sous-shells.

**Exemples :**

**Portée locale/exportée :**

```
v1=haha
export v2=hehe
bash           # Ouvre un sous-shell
echo $v1       # Rien (locale au parent)
echo $v2       # hehe (exportée)
exit           # Retour au parent
```

**Script avec `./` vs `source` :**

```
# test-shell
#!/bin/bash
echo $v1 $v2
```

`./test-shell` : Rien (sous-shell, variables non accessibles).
`source ./test-shell` : Affiche `haha hehe` (shell courant).

**Persistance :**Pour rendre des variables permanentes, ajoutez-les à `~/.bashrc` :

```
echo "export v2=hehe" >> ~/.bashrcsource ~/.bashrc
```

### 2.3. Scripting avancé

Création de scripts avec logique.

**Exemple : Vérification de quota (`test-quota`) :**

```
#!/bin/bash
MAX_USAGE="5G"
CURRENT_USAGE=$(du -sh ~ | cut -f1)  # Taille en format lisible
if (( $(echo "$CURRENT_USAGE < $MAX_USAGE" | bc -l) )); then
echo "Tout est ok."
exit 0
else
  echo "Pas assez d'espace !"
exit 1
fi
```

`bc -l` : Calcule une comparaison numérique (ex. `5G < 5G`).

### 2.4. Options de `set`

La commande `set` modifie le comportement de Bash.

**Options utiles :**

`set -f` : Désactive le globbing (expansion des `*`).

`set +f` : Réactive le globbing.

`set -C` : Empêche l'écrasement avec `>` (échoue si fichier existe).

`set -u` : Erreur si variable non définie.

**Exemples :**

**Globbing :**

```
bash
ls /bin/*      # Liste les fichiersset -fls /bin/*      # Cherche littéralement "/bin/*"set +f
# Réactive
```

**Écrasement :**

```
bash
set -Cecho "toto" > fich.txt  # Échoue si fich.txt existe
```

# Partie 3 : Processus, Daemons et Planification (TP3)

### 3.1. Processus et Daemons
Les processus sont des programmes en exécution, et les daemons fonctionnent en arrière-plan.

**Commandes :**
`ps` : Liste les processus (`-o %p%P%c` pour PID, PPID, nom).
`kill` : Envoie un signal (ex. `STOP`, `CONT`, terminaison).

**Exemples :**
**Processus enfants :**

```
ps -o %p%P%c  # Affiche PID, PPID, commande
```

**Contrôle d'un processus :**

```
xcalc &           # Lance en arrière-plan
kill -STOP <PID>  # Suspend
kill -CONT <PID>  # Reprend
kill <PID>        # Termine
```

**Daemons :**
Exemple : `cron` exécute des tâches planifiées.
Documentation : `man 7 daemon`.

### 3.2. Planification avec `at`
`at` planifie une tâche unique.

**Syntaxe :**
`at <heure> -f <script>` : Exécute un script à l'heure donnée.
Temps : `now + 3 minutes`, `midnight`, `5am tomorrow`.

**Exemples :**

**Dans 1 minute :**

```
at now + 1 minute -f ./script.sh
```

**À minuit :**

```
at midnight -f ./script.sh
```

**Fichiers de contrôle :**
`/etc/at.allow` : Utilisateurs autorisés.
`/etc/at.deny` : Utilisateurs interdits.

### 3.3. Boucles et conditions
Les boucles et la commande `test` permettent des scripts dynamiques.
**Commande `test` :**
`-e` : Existe.
`-x` : Exécutable.
`-n` : Chaîne non vide.
`-z` : Chaîne vide.
`-gt` : Supérieur.

**Exemple de boucle (`fich-doc`) :**

```
#!/bin/bashwhile [ $# -gt 0 ]; do  if [ -x "$1" ]; then    echo "$1 est accessible."  else
echo "$1 n'est pas accessible."  fi  shiftdone
```

# Partie 4 : Conditionnels et Crontab (TP4)

### 4.1. Structures conditionnelles
Les conditions contrôlent le flux d'exécution.

**Syntaxe `if` :**

```
if condition; then  # commandes
elif condition; then  # commandes
else
 # commandes
fi
```

**Exemple (`version-if`) :**

```
#!/bin/bash
if test -w "$1" -a -n "$2"; then
  echo "$2" >> "$1"
elif test -e "$1" -a ! -w "$1"; then
exit 1
elif test ! -e "$1" -o -z "$2"; then
exit 2
fi
```
Ajoute `$2` à `$1` si writable et `$2` non vide, sinon erreur spécifique.

### 4.2. Planification avec `crontab`
`crontab` planifie des tâches récurrentes.

**Commandes :**
`crontab -e` : Éditer.
`crontab -l` : Afficher.
`crontab -r` : Supprimer.

**Syntaxe :**
```
minute heure jour mois jour_semaine commande
```
`*` : Toutes les valeurs.

**Exemples :**

**Toutes les minutes :**

```
* * * * * ./script.sh
```
**1er du mois à midi :**

```
0 12 1 * * ./script.sh
```

**Fichiers de contrôle :**
`/etc/cron.allow` : Autorités.
`/etc/cron.deny` : Interdits.

# Questions du CTP

### Exercice 1.1 : Manipulations de paramètres

**Q1 : Commande** `dernier`
Affiche le dernier paramètre reçu, échoue si aucun paramètre.

**Solution :**

```
#!/bin/bash
if [ $# -eq 0 ]; then
exit 1
fi
echo "${!#}"  # ${!#} accède au dernier argument
```

**Explication :** `$#` vérifie le nombre d'arguments. `${!#}` est une syntaxe spéciale pour le dernier paramètre.

**Q2 : Commande** `options-legales`
Vérifie si toutes les options (commençant par `-`) sont dans `/home/public/r401/2024-2025/options-autorisees`.
**Solution :**

```
#!/bin/bash
for arg in "$@"; do
if [[ "$arg" =~ ^- ]] && ! grep -q "^$arg$" /home/public/r401/2024-2025/options-autorisees; then
exit 1
fi
done
exit 0
```

**Explication :** `=~ ^-` teste si l'argument commence par `-`. `grep -q` cherche l'option exacte dans le fichier.
**Exercice 1.2 : Histogramme**
**Q3 : Commande** `repeter`
Prend un entier `n` et une chaîne, affiche la chaîne `n` fois sur une ligne.
**Solution :**

```
#!/bin/bash
if [ $# -ne 2 ] || ! [[ "$1" =~ ^[0-9]+$ ]]; then
exit 1
fi
for ((i=0; i<$1; i++)); do
 echo -n "$2 "  # -n évite le retour à la lignedoneecho  # Nouvelle ligne finale
```

**Explication :** Vérifie 2 arguments et que `$1` est un entier. Boucle `for` pour répéter.
**Q4 : Commande** `statistique`
Affiche une ligne de `#` pour chaque entier en paramètre, en utilisant `repeter`.
**Solution :**

```
#!/bin/bash
for num in "$@"; do
if [[ "$num" =~ ^[0-9]+$ ]]; then
   ./repeter "$num" "#"
fi
done
```

**Explication :** Filtre les entiers avec une regex, puis appelle `repeter`.
**Exercice 2.1 : Fichiers PDF**
**Q5 : Commande** `fichier-pdf`
Vérifie si un fichier commence par `%PDF`, affiche son nom et réussit si vrai.
**Solution :**

```
#!/bin/bash
if [ $# -ne 1 ] || ! [ -f "$1" ]; then
exit 1
fi
if head -c 4 "$1" | grep -q "%PDF"; then
echo "$1"
exit 0
else
exit 1
fi
```

**Explication :** `head -c 4` lit les 4 premiers octets. `grep -q` teste `%PDF` silencieusement.

# GEOMETRY BASH, LE QUIZ

Instructions
Pour chaque question, choisissez la bonne réponse parmi les quatre options proposées (a, b, c, d).
Notez vos réponses sur une feuille ou mentalement, puis consultez la page de correction à la fin pour vérifier vos résultats.

Question 1
Quelle est la différence entre > et >> en Bash ?
a) > crée un nouveau fichier, >> ajoute à un fichier existant
b) > ajoute à un fichier existant, >> crée un nouveau fichier
c) > redirige la sortie standard, >> redirige la sortie d'erreur
d) > redirige la sortie d'erreur, >> redirige la sortie standard

Question 2
Comment définit-on une variable en Bash ?
a) var = valeur
b) var=valeur
c) set var=valeur
d) export var=valeur

Question 3
Que fait l'opérateur && en Bash ?
a) Exécute la commande suivante si la précédente échoue
b) Exécute la commande suivante si la précédente réussit
c) Concatène deux chaînes
d) Compare deux nombres

Question 4
Quelle est la première ligne d'un script Bash ?
a) #!/bin/bash
b) #bash
c) //bin/bash
d) /* bash */

Question 5
Quelle commande liste les processus en cours ?
a) ls
b) ps
c) top
d) kill

Question 6
Qu'est-ce qu'un daemon ?
a) Un processus qui s'exécute en arrière-plan
b) Un type de variable
c) Une commande de Bash
d) Un fichier de configuration

Question 7
Comment planifie-t-on une tâche avec at pour dans 5 minutes ?
a) at now + 5 minutes
b) at 5 minutes
c) at now 5 minutes
d) at +5 minutes

Question 8
Comment écrit-on une boucle for en Bash ?
a) for i in 1 2 3; do echo $i; done
b) for (i=1; i<=3; i++) { echo $i; }
c) for i from 1 to 3; echo $i; end
d) loop i=1 to 3; echo $i; endloop

Question 9
Quelle commande teste si un fichier existe ?
a) test -e fichier
b) exists fichier
c) file -e fichier
d) check fichier

Question 10
Comment planifie-t-on une tâche toutes les minutes avec crontab ?
a) * * * * * commande
b) 0 * * * * commande
c) * 0 * * * commande
d) 0 0 * * * commande

# TERMINE, GRO !!

## Regardons les réponses.

Question 1

Réponse correcte : a) > crée un nouveau fichier, >> ajoute à un fichier existant
Explication : En Bash, l'opérateur > redirige la sortie standard (stdout) vers un fichier. S'il existe déjà, il est écrasé ; sinon, un nouveau fichier est créé. L'opérateur >> redirige également la sortie standard, mais au lieu d'écraser le fichier, il ajoute la sortie à la fin du fichier existant. Par exemple : echo "texte" > fichier remplace le contenu de fichier, tandis que echo "texte" >> fichier ajoute "texte" à la suite.

Question 2
Réponse correcte : b) var=valeur
Explication : En Bash, une variable est définie avec la syntaxe nom=valeur, sans espaces autour du signe =. Par exemple, x=10 définit la variable x avec la valeur 10. L'option export (comme dans d) rend la variable accessible aux sous-processus, mais n'est pas nécessaire pour une simple définition.

Question 3
Réponse correcte : b) Exécute la commande suivante si la précédente réussit
Explication : L'opérateur && est un ET logique en Bash. Il exécute la commande suivante uniquement si la commande précédente retourne un code de sortie 0 (succès). Par exemple, cmd1 && cmd2 exécute cmd2 seulement si cmd1 réussit.

Question 4
Réponse correcte : a) #!/bin/bash
Explication : La première ligne d’un script Bash, appelée "shebang" ou "hashbang", est #!/bin/bash. Elle indique au système d'utiliser l'interpréteur Bash situé à /bin/bash pour exécuter le script. Les autres options ne sont pas des syntaxes valides.

Question 5
Réponse correcte : b) ps
Explication : La commande ps (process status) affiche les processus en cours d'exécution. Par exemple, ps aux liste tous les processus avec des détails. top fournit une vue dynamique, mais ps est la réponse la plus directe ici. ls liste des fichiers, et kill termine des processus.

Question 6
Réponse correcte : a) Un processus qui s'exécute en arrière-plan
Explication : Un daemon est un programme qui s'exécute en arrière-plan (background) pour fournir des services, comme un serveur web ou un gestionnaire de tâches planifiées. Il ne s'agit ni d'une variable, ni d'une commande, ni d'un fichier de configuration.

Question 7
Réponse correcte : a) at now + 5 minutes
Explication : La commande at planifie une tâche à exécuter à un moment précis. La syntaxe at now + 5 minutes indique que la tâche doit être exécutée 5 minutes à partir de l'heure actuelle. Les autres options ne sont pas des syntaxes valides pour at.

Question 8
Réponse correcte : a) for i in 1 2 3; do echo $i; done
Explication : En Bash, une boucle for utilise la syntaxe for variable in liste; do commandes; done. Ici, for i in 1 2 3; do echo $i; done affiche successivement 1, 2 et 3. Les autres options utilisent des syntaxes propres à d'autres langages (comme C ou pseudo-code) et ne fonctionnent pas en Bash.

Question 9
Réponse correcte : a) test -e fichier
Explication : La commande test -e fichier vérifie si le fichier spécifié existe. Elle est souvent utilisée dans des scripts avec une condition, comme if test -e fichier; then echo "existe"; fi. Les autres options ne sont pas des commandes Bash valides.

Question 10
Réponse correcte : a) * * * * * commande
Explication : Dans un fichier crontab, les cinq champs (* * * * *) représentent respectivement les minutes, heures, jours du mois, mois et jours de la semaine. Utiliser * pour chaque champ signifie "toutes les valeurs possibles", donc * * * * * commande exécute la commande toutes les minutes.


UX
BILL MOBIL

# 1. Introduction à Android Studio et aux concepts de base

Android Studio est l'IDE officiel pour développer des applications Android. Il inclut des outils pour concevoir des interfaces, coder, tester et déployer vos projets.

**Structure d'un projet Android** :

`app/src/main/java` : contient vos fichiers Java (ex. les activités).

`app/src/main/res` : regroupe les ressources comme les layouts XML, images, etc.

`AndroidManifest.xml` : fichier clé qui déclare les activités, permissions, etc.

**Créer une activité simple en Java** :

Une activité représente un écran. Voici un exemple de base :

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState)
{        super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main); // Lie le layout XML à l'activité    }}
```

# 2. Activités et disposition des vues

Les activités sont les écrans de l'application, et les vues (boutons, textes, etc.) sont organisées dans des layouts XML.

**Types de layouts courants** :

**LinearLayout** : aligne les éléments en ligne (horizontal ou vertical).

**ConstraintLayout** : positionnement flexible avec des contraintes.

**Exemple : Layout pour un écran simple** :

Un layout avec un titre et un bouton :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"    android:layout_height="match_parent"
android:orientation="vertical">    <TextView        android:layout_width="wrap_content"
android:layout_height="wrap_content"        android:text="Bienvenue" />    <Button
android:id="@+id/btn_click_me"        android:layout_width="wrap_content"
android:layout_height="wrap_content"        android:text="Cliquez-moi" /></LinearLayout>
```

**Interagir avec les vues en Java** :

Utilisez `findViewById` pour accéder aux éléments du layout :

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
private Button btnClickMe;
@Override    protected void onCreate(Bundle savedInstanceState)
{        super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
btnClickMe = findViewById(R.id.btn_click_me);
btnClickMe.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
btnClickMe.setText("Vous avez cliqué !");
 }        });    }}
```

**3. Navigation entre activités**
Pour passer d'un écran à un autre, on utilise des **Intents**.
**Exemple : Lancer une nouvelle activité** :

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
private Button btnClickMe;
@Override
protected void onCreate(Bundle savedInstanceState)
{        super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
btnClickMe = findViewById(R.id.btn_click_me);
btnClickMe.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(MainActivity.this, SecondActivity.class);
startActivity(intent);            }        });    }}
```

**Cycle de vie d'une activité** :
Les méthodes clés incluent `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroy()`.
Exemple d'utilisation :

```
@Override
protected void onResume() {
super.onResume();    // Mettre à jour l'interface quand l'activité redevient visible}
```

**4. Afficher une liste avec RecyclerView**
Le `RecyclerView` est idéal pour afficher des listes dynamiques.
**Composants** :
**RecyclerView** : conteneur de la liste.
**Adapter** : relie les données aux vues.
**ViewHolder** : gère les éléments d'un item.
**Exemple : Layout d'un item** :

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"    android:layout_height="wrap_content">    <TextView
android:id="@+id/item_text"        android:layout_width="wrap_content"
android:layout_height="wrap_content" /></LinearLayout>
```

**Code Java pour RecyclerView** :

Regarde CTP24

ou https://gitlab.univ-lille.fr/nils.vanderschooten-akinmoladun.etu/devmob_ctp_2024

# Pour Retrouver un exemple d'utilisation des concepts

# 5. Menus et fragments

**Ajouter un menu** :

Créez `res/menu/menu_main.xml` :

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">    <item
android:id="@+id/action_settings" android:title="Paramètres" /></menu>
```

Puis dans l'activité :

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{    getMenuInflater().inflate(R.menu.menu_main, menu);    return true;}
```

**Fragments** :

Exemple simple :

```
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.fragment.app.Fragment;
public class MyFragment extends Fragment {
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
 return inflater.inflate(R.layout.fragment_layout, container, false);    }}
```

# PILE OU MOBILE
# LE TEST

Instructions

- Choisissez la bonne réponse (a, b, c ou d) pour chaque question.
- Notez vos réponses, puis consultez la section des corrections à la fin.

Question 1

Quel est le rôle du fichier AndroidManifest.xml dans un projet Android ?

a) Il contient le code Java de l'activité principale

b) Il définit la mise en page de l'interface utilisateur

c) Il déclare les activités, permissions et autres composants de l'application

d) Il stocke les ressources de l'application comme les images et les chaînes

Question 2

Quel layout utiliseriez-vous pour aligner des vues horizontalement en une rangée ?

a) ConstraintLayout

b) LinearLayout avec une orientation verticale

c) LinearLayout avec une orientation horizontale

d) RelativeLayout

Question 3

Comment démarre-t-on une nouvelle activité depuis l'activité actuelle en Java ?

a) startActivity(new Activity());

b) Intent intent = new Intent(this, NewActivity.class); startActivity(intent);

c) startNewActivity(NewActivity.class);

d) Intent intent = new Intent(); startActivity(intent);

Question 4

Quel est le rôle de l'Adapter dans un RecyclerView ?

a) Il gère la mise en page des éléments de la liste

b) Il contient les données et crée les vues pour chaque élément

c) Il gère les interactions de l'utilisateur avec la liste

d) Il définit la mise en page XML du RecyclerView

Question 5

Comment gonfle-t-on (inflate) un menu dans une activité ?

a) getMenuInflater().inflate(R.menu.menu_main, menu);

b) inflateMenu(R.menu.menu_main);

c) menu.inflate(R.menu.menu_main);

d) MenuInflater.inflate(R.menu.menu_main, menu);

Question 6

Quelle méthode surcharge-t-on pour définir la mise en page d'un fragment ?

a) onCreate()

b) onCreateView()

c) onAttach()

d) onViewCreated()

Question 7



Quelle est une bonne méthode pour persister de petites quantités de données, comme les préférences utilisateur ?

a) Utiliser une base de données SQLite

b) Écrire dans un fichier en stockage interne

c) Utiliser SharedPreferences

d) Stocker dans une variable statique

# PILE OU MOBILE
# LE TEST

Question 1
Réponse correcte : c) Il déclare les activités, permissions et autres composants de l'application
Explication : Le fichier AndroidManifest.xml est essentiel pour définir la structure de votre application. Il répertorie toutes les activités, services, permissions et autres composants qui composent l'application. Les options a, b et d sont incorrectes car le manifeste ne contient pas de code Java, de mises en page ou de ressources comme des images.

Question 2
Réponse correcte : c) LinearLayout avec une orientation horizontale
Explication : Un LinearLayout avec une orientation horizontale dispose ses vues enfants en une seule rangée. L'option a (ConstraintLayout) est plus flexible mais pas spécifique aux rangées. L'option b empile les vues verticalement, et l'option d (RelativeLayout) positionne les vues relativement les unes aux autres, pas nécessairement en rangée.

Question 3
Réponse correcte : b) Intent intent = new Intent(this, NewActivity.class); startActivity(intent);
Explication : C'est la méthode standard pour créer un Intent et démarrer une nouvelle activité dans Android avec Java. L'option a est incorrecte car on ne peut pas instancier directement une Activity. L'option c n'est pas une méthode valide, et l'option d manque la classe de l'activité cible.

Question 4
Réponse correcte : b) Il contient les données et crée les vues pour chaque élément
Explication : L'Adapter dans un RecyclerView est responsable de fournir les vues qui représentent chaque élément de l'ensemble de données. Il gère les données et crée des objets ViewHolder pour un recyclage efficace des vues. Les options a, c et d décrivent d'autres composants ou sont incorrectes.

Question 5
Réponse correcte : a) getMenuInflater().inflate(R.menu.menu_main, menu);
Explication : C'est la manière correcte de gonfler un menu dans la méthode onCreateOptionsMenu() d'une activité. Les options b et c ne sont pas des méthodes valides, et l'option d utilise incorrectement MenuInflater sans contexte.

Question 6
Réponse correcte : b) onCreateView()
Explication : Dans un fragment, onCreateView() est la méthode où vous gonflez et retournez la mise en page de l'interface utilisateur du fragment. L'option a est pour l'initialisation, l'option c pour l'attachement à l'activité, et l'option d est appelée après la création de la vue.

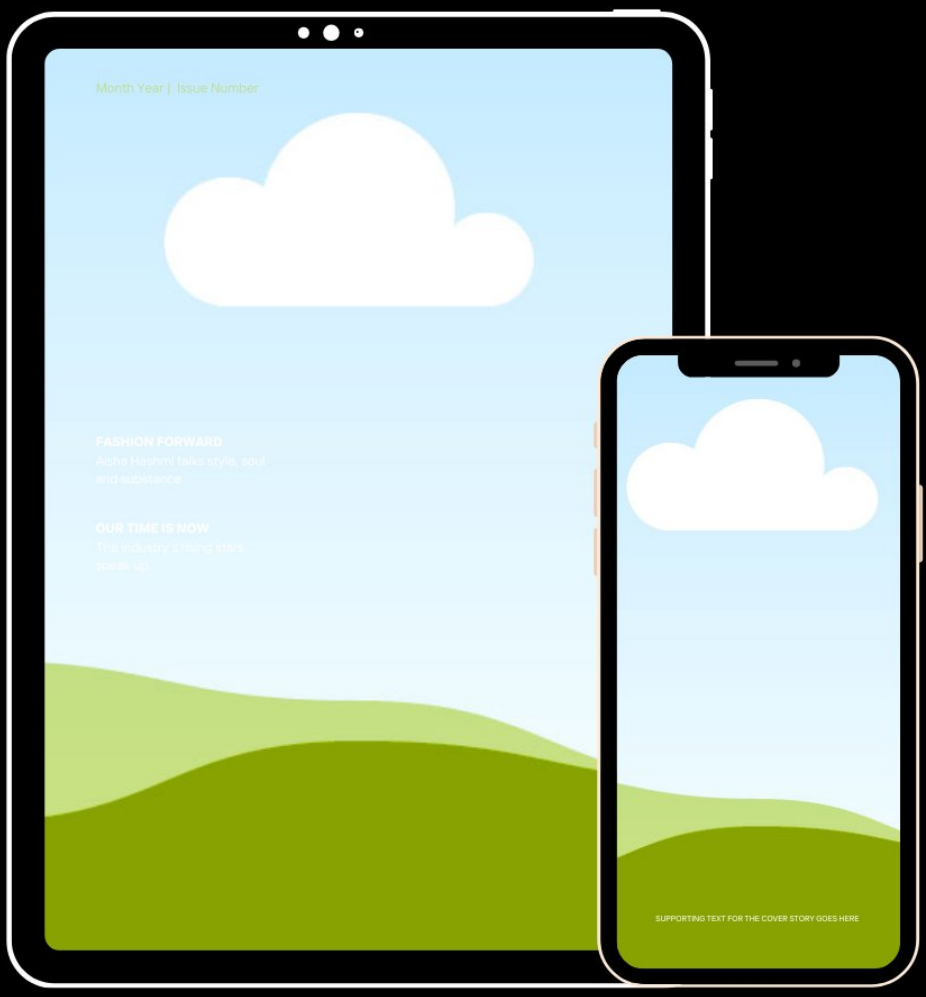
Question 7
Réponse correcte : c) Utiliser SharedPreferences
Explication : SharedPreferences est idéal pour stocker de petites quantités de données, comme les préférences utilisateur, sous forme de paires clé-valeur. L'option a (SQLite) est excessive pour de petites données, l'option b est moins efficace, et l'option d (variable statique) ne persiste pas les données après un redémarrage de l'application.

www.intelli-odoo.com

Intelli et

# INTELLISOCIAL

Le futur réseau social, le 19 Mars.




**Je sais plus quoi marquer**
feur

**A Mardi prochain !**