GUIDE DES MEILLEURES PRATIQUES EN COBOL

FromZeroToCobol

Bienvenue dans l'Univers COBOL

Cher Apprenant,

Vous tenez entre vos mains (ou sur votre écran) le passeport vers l'excellence en **COBOL**. *Ce guide a été conçu spécialement pour vous, que vous soyez débutant enthousiaste ou développeur expérimenté cherchant à aiguiser encore davantage vos compétences. Le langage* **COBOL**, *pierre angulaire des systèmes informatiques grands systèmes depuis des décennies, continue de jouer un rôle crucial dans le monde de la programmation moderne. Sa maîtrise est non seulement un atout précieux pour votre carrière, mais aussi une porte ouverte sur l'entretien et l'innovation des systèmes financiers, bancaires, d'assurances et bien d'autres qui soutiennent notre économie globale.*

Dans ce guide, nous parcourons ensemble les meilleures pratiques qui vous mèneront vers un code **COBOL** *efficace, maintenable et optimisé. À travers ces pages, vous découvrirez les fondements de la programmation* **COBOL**, *enrichis de conseils pratiques, de stratégies éprouvées et de techniques de débogage qui transformeront votre approche du code.*

Au programme:

Introduction

- Brève introduction au **COBOL** et son importance dans le monde informatique moderne.
- Objectif du guide et comment il peut aider les développeurs à écrire un code **COBOL** plus efficace et maintenable.

Chapitre 1: Fondations du COBOL

- Comprendre la Syntaxe **COBOL** : Les bases de la syntaxe pour écrire des instructions claires et lisibles.
- Structure d'un Programme **COBOL** : Vue d'ensemble des quatre divisions principales et leur rôle.

Chapitre 2: Bonnes Pratiques de Codage

- Conventions de Nommage : Importance de noms descriptifs pour les variables, les fichiers, et les routines.
- Commentaires et Documentation : Comment bien commenter votre code pour une meilleure compréhension et maintenance.

Chapitre 3: Gestion Efficace des Données

- Déclarations de Données : Bonnes pratiques pour déclarer et structurer les données.
- Manipulation des Fichiers : Conseils pour une manipulation efficace des fichiers.

Chapitre 4: Améliorer la Lisibilité et la Maintenance

- Structure de Contrôle : Utilisation efficace des instructions conditionnelles et des boucles pour un code plus propre.
- Modularité et Réutilisabilité : Comment structurer votre code pour faciliter la réutilisation et la maintenance.

Chapitre 5: Optimisation et Performance

- Techniques d'Optimisation : Conseils pour améliorer la performance de vos programmes **COBOL**.
- Gestion des Erreurs : Stratégies pour une gestion d'erreur robuste et préventive.

Chapitre 6: Se Préparer pour l'Avenir

- **COBOL** dans le Contexte Moderne : L'intégration de **COBOL** avec les technologies modernes et les systèmes d'exploitation.
- Développement Continu : Importance de la formation continue et comment rester à jour avec les tendances **COBOL**.

Chapitre Bonus: COBOL en Pêle-Mêle : Bonnes Pratiques et Conseils

Conclusion

- Récapitulatif des meilleures pratiques en **COBOL**
- Ressources Complémentaires

Introduction

Bienvenue dans ce voyage à la découverte des meilleures pratiques en **COBOL**, un guide essentiel pour façonner un avenir prometteur dans le domaine de la programmation grands systèmes. Ici, vous trouverez un condensé de connaissances, des astuces de pro et des recommandations pour écrire un code non seulement performant mais aussi facile à entretenir et à comprendre. Cet outil est votre allié pour transcender les bases du **COBOL** et explorer des techniques avancées, vous préparant ainsi à contribuer de manière significative au paysage technologique actuel et futur. Embarquez avec confiance dans cette aventure d'apprentissage qui élèvera vos compétences en **COBOL** à un niveau d'expertise recherché.

Dans ce contexte, l'importance du **COBOL** ne peut être sous-estimée. Ce langage, qui a façonné l'infrastructure informatique mondiale depuis plus d'un demi-siècle, continue d'être au cœur de systèmes critiques dans presque tous les secteurs. Des transactions bancaires quotidiennes aux opérations d'assurance, en passant par la gestion des systèmes de santé et de retraite, **COBOL** gère des milliards

de lignes de code qui soutiennent notre économie et notre société. Sa robustesse, sa fiabilité et sa capacité à traiter de vastes volumes de données le rendent indispensable dans le paysage technologique moderne. Ainsi, maîtriser le **COBOL** n'est pas seulement un investissement dans votre carrière, mais aussi une contribution précieuse à la maintenance et à l'évolution de systèmes vitaux pour notre quotidien. En approfondissant vos connaissances et compétences en **COBOL**, vous vous positionnez comme un acteur clé capable de naviguer et d'innover dans cet écosystème essentiel.

Votre engagement dans l'apprentissage du **COBOL** ouvre la porte à des opportunités uniques dans le monde de la technologie. Alors que de nombreux secteurs recherchent constamment des innovations et des améliorations, le rôle du **COBOL** demeure fondamental, assurant la stabilité et la fiabilité nécessaires à leur fonctionnement. Cet héritage, combiné à une demande persistante pour des compétences spécialisées en **COBOL**, souligne non seulement l'importance mais aussi la pertinence continue de ce langage dans l'ère numérique.

1. Fondations du COBOL

Découvrons ensemble les fondements du **COBOL**, un langage structuré autour de la clarté et de la précision. Sa syntaxe, pensée pour être aisément lisible, et sa structure de programme, divisée en sections bien définies, constituent le cœur de l'apprentissage de ce langage puissant.

Comprendre la Syntaxe COBOL

La syntaxe du **COBOL**, conçue pour la lisibilité, emploie des mots anglais complets et des phrases structurées. Ce choix rend le **COBOL** unique et accessible.

Exemple de Syntaxe

Imaginons que nous voulons assigner une valeur à une variable. En **COBOL**, cela ressemblerait à :

```
MOVE 100 TO WS-TOTAL-SALES.
```

Cet exemple montre comment une instruction **COBOL** peut être intuitive, comme lire une phrase en anglais, où nous "**déplaçons**" la valeur 100 dans la variable **WS-TOTAL-SALES**.

1. Fondations du COBOL

Structure d'un Programme COBOL

Chaque programme **COBOL** est méthodiquement organisé en quatre divisions principales, chacune ayant un rôle spécifique dans la définition et le fonctionnement du programme.

Division IDENTIFICATION

Elle sert de **préambule au programme**, offrant des informations essentielles telles que le nom du programme et le nom de l'auteur. C'est la carte d'identité du programme.

Division ENVIRONMENT

Cette division configure **l'environnement matériel** et logiciel, précisant où et comment le programme sera exécuté. Elle peut par exemple définir le type d'ordinateur et le système d'exploitation cible.

Division DATA

Ici, on déclare toutes les structures de données : les fichiers à traiter, les variables à utiliser, etc. C'est le cœur de la gestion des données dans un programme **COBOL**.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-TOTAL-SALES PIC 9(3) VALUE 0.
```

Ce fragment déclare une variable **WS-TOTAL-SALES** capable de stocker **un nombre jusqu'à 999, initialisée à 0**.

Division PROCEDURE

Correspond à la dernière division, où le véritable traitement des données a lieu. Ici, nous écrivons le code qui effectue des **opérations sur les données, comme des calculs, des boucles, et des conditions**.

Exemple de Code dans la PROCEDURE DIVISION

```
PROCEDURE DIVISION.
START-PROGRAM.
    ADD 50 TO WS-TOTAL-SALES.
    DISPLAY 'Total Sales: ' WS-TOTAL-SALES.
```

Cet exemple simple **ajoute 50 à notre variable WS-TOTAL-SALES** et **affiche le résultat**.

En assimilant ces concepts de syntaxe et de structure, vous posez une fondation solide pour votre voyage dans le monde du **COBOL**. Ces éléments sont les premiers pas vers l'écriture de programmes **COBOL** efficaces, maintenables et optimisés, vous préparant à explorer plus en profondeur les techniques avancées dans les prochains chapitres.

2. Bonnes Pratiques de Codage

Après avoir exploré les fondamentaux du **COBOL**, concentrons-nous sur l'amélioration de la qualité de notre code à travers des pratiques de codage éprouvées. Une bonne pratique de codage ne se limite pas à suivre les règles syntaxiques ; elle englobe également la manière dont nous nommons et documentons notre code pour garantir sa clarté, sa maintenabilité et sa facilité de compréhension.

Conventions de Nomination

Les conventions de nomination jouent un rôle crucial dans la lisibilité du code. Adopter des conventions cohérentes et descriptives aide les développeurs à comprendre rapidement la fonction et le but des variables et des sections de travail.

Exemples de Variables et Sections de Travail

- **Variables en Majuscules avec Préfixe** : En **COBOL**, il est courant de voir des variables de la section **WORKING-STORAGE** commencer par **WS-** et être écrites en majuscules pour une identification rapide.

```
WORKING-STORAGE SECTION.
01 WS-CUSTOMER-NAME    PIC A(30).
01 WS-TOTAL-AMOUNT     PIC 9(5)V99.
```

Dans cet exemple, **WS-CUSTOMER-NAME** est une variable destinée à **stocker un nom de client**, tandis que **WS-TOTAL-AMOUNT** représente un **montant total**, avec une précision de deux chiffres après la virgule.

Commentaires et Documentation

Une documentation cohérente et des commentaires pertinents sont essentiels pour que le code soit auto-explicatif et facile à maintenir.

Intégration de Commentaires Pertinents

- **Clarification du Code** : Les commentaires doivent fournir une **valeur ajoutée** en expliquant la raison d'être du code, surtout pour les parties complexes ou non intuitives.

```
* Ce calcul ajuste le total en fonction de la remise client
COMPUTE WS-TOTAL-AMOUNT = WS-TOTAL-AMOUNT - WS-DISCOUNT.
```

- **En-tête de Programme** : Commencez chaque programme avec un **en-tête détaillé pour offrir un contexte clair sur le programme**, son auteur, et son but.

2. Bonnes Pratiques de Codage

```
*-------------------------------------------------------
* Nom du Programme : COBOL001
* Description : Ce programme calcule les remises clients.
* Auteur          : NICOLAS
* Date Création    : 01/01/2021
* Modifications    : N/A
*-------------------------------------------------------
```

En respectant ces bonnes pratiques de codage, vous vous assurez non seulement de produire des programmes **COBOL** efficaces mais aussi maintenables et facilement compréhensibles par d'autres. Les conventions de nomination précises, associées à une documentation et des commentaires adéquats, forment la colonne vertébrale d'un code de qualité, préparant ainsi le terrain pour une maintenance aisée et une collaboration fructueuse au sein de votre équipe de développement.

3. Gestion Efficace des Données

Une gestion efficace des données est essentielle dans tout programme **COBOL**. Ce chapitre se concentre sur les meilleures pratiques pour déclarer et structurer vos données, ainsi que sur des conseils pour manipuler efficacement les fichiers. Ces pratiques sont cruciales pour garantir que vos programmes sont non seulement performants mais aussi maintenables et évolutifs.

Déclarations de Données

En **COBOL**, la déclaration des données est effectuée dans la **DATA DIVISION**, qui est subdivisée en plusieurs sections, chacune ayant un objectif spécifique. La clarté et la précision dans cette division sont vitales.

Exemple de Déclaration Structurée

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 WS-EMPLOYEE-RECORD.
   05 WS-EMPLOYEE-ID          PIC 9(5).
   05 WS-EMPLOYEE-NAME        PIC A(20).
   05 WS-EMPLOYEE-SALARY      PIC 9(5)V99.
```

3. Gestion Efficace des Données

Dans cet exemple, **WS-EMPLOYEE-RECORD** est une structure de données composite contenant **l'ID** de **l'employé**, le **nom** et le **salaire**. La **structuration claire** et l'utilisation du **préfixe WS-** pour les variables de la **WORKING** facilitent la compréhension de leur usage.

Manipulation des Fichiers

La manipulation efficace des fichiers est fondamentale en **COBOL**, étant donné son utilisation étendue dans les applications de traitement de données en volume.

Lecture et Écriture de Fichiers

L'organisation et l'accès aux fichiers doivent être soigneusement planifiés. Voici un exemple de manipulation de fichier basique :

```
FD  EMPLOYEE-FILE
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS EMPLOYEE-RECORD.
01  EMPLOYEE-RECORD PIC X(80).
SELECT EMPLOYEE-FILE ASSIGN TO INP001.
```

Ce fragment de code définit un fichier EMPLOYEE-FILE et associe une structure de données EMPLOYEE-RECORD

pour la lecture ou l'écriture. Cela illustre l'importance de **nommer clairement les fichiers et les structures de données pour une gestion efficace**.

Bonnes Pratiques

- **Utilisation des Sections Appropriées** : Placez les déclarations de fichiers dans la **FILE SECTION** et définissez les structures de données correspondantes dans la **WORKING-STORAGE SECTION** ou la **LINKAGE SECTION** pour les données passées entre les programmes.

- **Accès Séquentiel ou Aléatoire** : Choisissez l'accès aux fichiers (**séquentiel, aléatoire ou dynamique**) basé sur les besoins opérationnels, optimisant ainsi les performances de lecture/écriture.

- **Gestion des Erreurs de Fichier** : Implémentez une **gestion d'erreur robuste** lors des opérations de fichier pour traiter les cas comme les lectures au-delà de la fin de fichier (EOF) ou les erreurs d'écriture.

3. Gestion Efficace des Données

En adhérant à ces principes de déclaration et de manipulation des données, vous assurez la robustesse et la fiabilité de vos programmes **COBOL**. La gestion efficace des données permet à vos applications de traiter des informations complexes de manière optimale.

4. Améliorer la Lisibilité et la Maintenance

Une fois les bases de la syntaxe, des conventions de nommage et de la gestion des données établies, il est crucial de se concentrer sur l'amélioration de la lisibilité et de la maintenance du code **COBOL**. Ce chapitre aborde l'importance des structures de contrôle bien conçues et la modularité pour faciliter la réutilisation et la maintenance du code.

Structure de Contrôle

Les structures de contrôle en **COBOL**, telles que les conditions **IF** et les boucles **PERFORM**, sont essentielles pour diriger le flux du programme. Leur utilisation efficace est primordiale pour maintenir le code propre et compréhensible.

Exemple d'Utilisation des Conditions

```
IF WS-AGE >= 18
   DISPLAY 'Vous êtes majeur.'
ELSE
   DISPLAY 'Vous êtes mineur.'
END-IF.
```

4. Améliorer la Lisibilité et la Maintenance

Cet exemple simple illustre l'utilisation d'une condition IF pour vérifier l'âge d'un individu. L'utilisation claire de l'espace et des indentations, ainsi que des conditions explicites, renforcent la lisibilité.

Exemple de Boucle PERFORM

```
PERFORM VARYING WS-COUNTER FROM 1 BY 1 UNTIL WS-COUNTER > 10
    DISPLAY 'Compteur : ' WS-COUNTER
END-PERFORM.
```

Cette boucle **PERFORM** affiche les nombres de 1 à 10, démontrant comment les boucles peuvent être utilisées pour répéter des actions de manière concise.

Modularité et Réutilisabilité

La structuration de votre code pour promouvoir la modularité et la réutilisabilité est une autre pierre angulaire des bonnes pratiques en **COBOL**. Diviser le code en modules ou en sections réutilisables peut grandement faciliter la maintenance et l'évolution des programmes.

4. Améliorer la Lisibilité et la Maintenance

Exemple de Modularité

Imaginons que nous avons une routine pour calculer le salaire net :

```
PROCEDURE DIVISION.
CALCULATE-NET-SALARY SECTION.
    MOVE WS-GROSS-SALARY TO WS-NET-SALARY.
    SUBTRACT WS-TAX FROM WS-NET-SALARY.
    SUBTRACT WS-SOCIAL-SECURITY
                                FROM WS-NET-SALARY.
```

En **isolant cette logique dans une section dédiée**, nous **facilitons sa réutilisation et sa maintenance**, en permettant également d'ajuster facilement les composantes du calcul du salaire net sans affecter d'autres parties du programme.

En appliquant ces principes d'utilisation efficace des structures de contrôle et en embrassant la modularité, vous préparez vos programmes **COBOL** à être plus accessibles, maintenables, et évolutifs. Ce chapitre fournit les clés pour structurer votre code de manière à encourager non seulement une meilleure compréhension immédiate mais aussi une adaptation facile aux besoins futurs.

5. Optimisation et Performance

L'optimisation et la performance sont des aspects critiques du développement en **COBOL**, surtout compte tenu de l'utilisation extensive de ce langage dans les applications d'entreprise où l'efficacité et la rapidité sont primordiales. Ce chapitre vous présente des techniques d'optimisation spécifiques au **COBOL** et des stratégies pour une gestion robuste des erreurs, garantissant ainsi des programmes non seulement performants mais aussi fiables.

Techniques d'Optimisation

Optimiser votre code **COBOL** pour une performance accrue implique plusieurs stratégies, allant de la gestion efficace des données à l'utilisation judicieuse des ressources systèmes.

Exemple d'Optimisation de Boucle

```
PERFORM VARYING WS-INDEX FROM 1 BY 1 UNTIL WS-INDEX > WS-MAX
    COMPUTE WS-TOTAL = WS-TOTAL + WS-ARRAY(WS-INDEX)
END-PERFORM.
```

5. Optimisation et Performance

Dans cet exemple, l'optimisation peut être réalisée en réévaluant la nécessité de parcourir chaque élément du tableau. Si l'objectif est de trouver la somme totale, **envisagez d'utiliser des techniques de réduction ou d'accumulation qui pourraient être plus efficaces**, ou assurez-vous que **WS-MAX** est ajusté au nombre minimum d'itérations nécessaires.

Gestion Efficace des Accès aux Fichiers

L'accès aux fichiers, notamment dans le cadre des opérations **d'entrée/sortie**, peut considérablement impacter la performance. Minimiser le nombre d'opérations **d'entrée/sortie** en regroupant les données ou en utilisant des tampons peut réduire le temps d'exécution.

```
OPEN INPUT EMPLOYEE-FILE.
READ EMPLOYEE-FILE INTO WS-EMPLOYEE-RECORD.
```

Pour optimiser cela, **envisagez de lire les données en blocs si votre logique métier le permet**.

Gestion des Erreurs

Une gestion d'erreur robuste et préventive est essentielle pour **maintenir la performance et la fiabilité des programmes COBOL**.

5. Optimisation et Performance

Stratégies de Gestion des Erreurs

- **Validation des Données** : Assurez-vous de valider les entrées pour éviter des erreurs d'exécution qui peuvent être coûteuses en termes de performance.

```
IF NOT NUMERIC WS-EMPLOYEE-ID
   DISPLAY "Erreur : ID Employé invalide."
END-IF.
```

- **Gestion des Erreurs d'Entrée/Sortie** : Traitez efficacement les erreurs d'entrée/sortie pour éviter des interruptions inattendues et garantir la continuité de l'application.

```
READ EMPLOYEE-FILE INTO WS-EMPLOYEE-RECORD
   AT END
      DISPLAY "Fin du fichier atteinte."
   NOT AT END
      CONTINUE
END-READ.
```

En mettant en œuvre ces techniques d'optimisation et en adoptant une approche proactive à la gestion des erreurs, vous pouvez améliorer significativement la performance et la fiabilité de vos applications **COBOL**. La clé est de rester vigilant et de tester régulièrement les performances.

6. Se Préparer pour l'Avenir

Le monde de la technologie évolue à une vitesse fulgurante, et avec lui, les langages de programmation et les environnements d'exploitation continuent de se transformer. Cependant, le **COBOL**, avec ses décennies de service fiable, **demeure un pilier dans de nombreux systèmes critiques**. Pour rester pertinent et efficace dans ce paysage en mutation, il est crucial de comprendre comment le **COBOL** s'intègre dans le contexte moderne et de s'engager dans un développement continu.

COBOL dans le Contexte Moderne

L'intégration du **COBOL** avec les **technologies modernes et les systèmes d'exploitation** n'est pas seulement possible ; elle est essentielle pour tirer parti des **avancées technologiques tout en maintenant les systèmes existants**.

Exemple d'Intégration avec des Systèmes Modernes

```
CALL 'API_REST' USING WS-URL WS-RESPONSE.
```

Dans cet exemple fictif, un programme **COBOL** utilise un appel **CALL** pour interagir avec une **API REST moderne**, illustrant comment le **COBOL** peut communiquer avec des services web externes.

6. Se Préparer pour l'Avenir

WS-URL contient **l'URL** de **l'API**, tandis que **WS-RESPONSE** est utilisé pour **stocker la réponse de l'API**. Ce type d'intégration permet aux applications **COBOL** de bénéficier de fonctionnalités web modernes tout en continuant à exécuter des opérations de base de données et de traitement en batch.

Développement Continu

L'importance de la formation continue ne peut être sous-estimée dans un domaine qui évolue rapidement. Pour les développeurs **COBOL**, cela signifie non seulement rester à jour avec les dernières versions du langage lui-même mais aussi comprendre comment intégrer des technologies émergentes.

Stratégies pour Rester à Jour

- **Participer à des Forums et des Communautés en Ligne** : Engagez-vous dans des forums dédiés au **COBOL** et à la programmation de systèmes grands systèmes pour échanger des connaissances et des expériences.
- **Suivre des Formations et des Webinaires** : Profitez des ressources de formation continues, des webinaires, et des ateliers pour découvrir les dernières pratiques et outils.

6. Se Préparer pour l'Avenir

- **Expérimenter avec des Projets Personnels** : Mettez en pratique vos connaissances en travaillant sur des projets personnels ou en contribuant à des projets open source, ce qui peut vous fournir une expérience précieuse avec de nouvelles technologies.

Se préparer pour l'avenir en tant que développeur **COBOL** signifie embrasser le changement et être prêt à apprendre et à s'adapter. En intégrant le **COBOL** avec des technologies modernes et en s'engageant dans un développement continu, vous pouvez non seulement prolonger la durée de vie des applications **COBOL** existantes mais aussi ouvrir de nouvelles voies pour l'innovation. Ce chapitre vous a armé avec les connaissances nécessaires pour naviguer dans le paysage technologique en évolution, assurant que vos compétences en **COBOL** restent précieuses et demandées.

COBOL en Pêle-Mêle : Bonnes Pratiques et Conseils

Avant de conclure cet ebook, nous vous proposons une dernière section dédiée à un **pêle-mêle de bonnes pratiques en COBOL**. Cette compilation de conseils et astuces vous **aidera à renforcer vos compétences et à adopter les meilleures approches dans vos projets**. Prenez le temps de parcourir ces suggestions et d'intégrer celles qui vous semblent les plus pertinentes pour optimiser votre travail en **COBOL**.

Dans la DATA DIVISION:

Utilisation du Mot Clé ZERO pour les Clauses VALUE

Pour les clauses « **VALUE** », utilisez le mot clé « **ZERO** » au lieu de '**0**'.

Exemple :

```
01 WS-NUM-COMMANDE.
   05 WS-STATUT-COMMANDE PIC 9 VALUE ZERO.
   88 WS-COMMANDE-COMPLETE VALUE 1.
```

Cela permet d'éviter les régressions lors des modifications de la taille des variables.

Alignement des Niveaux dans la WORKING et la LINKAGE

Pour améliorer la lisibilité de votre code :

- Alignez tous les éléments du même niveau sur la même colonne.
- Décalez les sous-niveaux de 3 colonnes par rapport à leur niveau supérieur.

Exemple :

```
01 WS-EMPLOYE.
   05 WS-NOM       PIC X(30).
   05 WS-AGE       PIC 99.
      10 WS-AGE-ANNEES PIC 9(2).
```

Utilisation du Mot Clé SPACE pour les Clauses VALUE

Pour les clauses « **VALUE** », utilisez le mot clé « **SPACE** » au lieu de l'espace vide (' ').

Exemple :

```
01 WS-DOCUMENT.
   05 WS-TITRE      PIC X(50) VALUE SPACE.
   05 WS-AUTEUR     PIC X(30) VALUE SPACE.
```

Cela prévient les régressions lors de l'extension de la taille de la variable.

Éviter l'Utilisation de la Clause OCCURS 1

Il est recommandé de ne pas déclarer de variable avec la clause « **OCCURS 1** ».

Exemple :

```
01 WS-TRANSACTIONS.
   05 WS-DATE-TRANS OCCURS 10 TIMES PIC 9(8).
```

Privilégiez des structures plus appropriées pour éviter la redondance et améliorer la lisibilité. La clause « **OCCURS** » est utilisée pour **définir des tableaux ou des structures répétitives**. Cependant, spécifier « **OCCURS 1** » n'apporte aucune valeur ajoutée et peut **rendre le code moins lisible et plus confus**.

Format des Indices de Tableaux

Pour **améliorer les performances et minimiser les risques de dépassement d'indice**, utilisez des **indices de tableaux au format COMP**. Ce format offre une meilleure performance et permet de gérer un grand nombre de positions.

Dans la PROCEDURE DIVISION:

Indentation des Blocs de Code

Pour une **meilleure lisibilité**, toutes les instructions après un **IF**, **EVALUATE**, ou **PERFORM** devraient être **indentées de trois espaces**. Cette règle s'applique surtout si les noms des variables sont longs, nécessitant parfois de scinder les instructions sur plusieurs lignes.

Exemple :

```
IF COMPTEUR = 0
  MOVE 0 TO RESULTAT
  ...
IF VILLE-CODE = 75
  MOVE 10 TO RESULTAT
  ...
```

Restreindre l'Imbrication des Structures de Contrôle

L'imbrication des structures IF doit être limitée à **cinq niveaux pour simplifier la compréhension et la maintenance du code**. Ce ne sont que de simples recommandations pour maintenir une compréhension optimale du code.

Utilisation d'un Seul Verbe COBOL par Instruction

Chaque ligne de code doit contenir **un seul verbe COBOL** pour assurer une clarté maximale.

Exemple à éviter:

```
EVALUATE TRUE
  WHEN AGE >= 18 MOVE 'ADULTE' TO STATUT
  WHEN OTHER MOVE 'MINEUR' TO STATUT
```

Exemple à privilégier:

```
EVALUATE TRUE
  WHEN AGE >= 18
    MOVE 'ADULTE' TO STATUT
  WHEN OTHER
    MOVE 'MINEUR' TO STATUT
```

Conventions de Fin d'Instructions

Les **instructions IF** doivent se terminer avec **END-IF**. De même, **PERFORM** et **EVALUATE** doivent se terminer par **END-PERFORM** et **END-EVALUATE**, respectivement, alignées avec la clause de début.

Alignement des Instructions

Alignez le début de chaque **instruction COBOL** sur la **colonne 12** pour **uniformiser la présentation** du code à travers le programme.

Exclusion du Code Mort

Éliminez tout code qui ne participe pas au déroulement normal du programme, comme celui qui ne peut jamais être exécuté ou qui représente des redondances conceptuelles.

Interdiction d'utilisation du COMPUTE sans Opération Arithmétique

Il est essentiel de **ne pas utiliser COMPUTE pour de simples affectations**. Utilisez **MOVE** pour ces opérations pour clarifier que aucun calcul n'est impliqué.

Exemple:

```
MOVE A TO B
```

Clause EVALUATE... WHEN OTHER

Toute structure **EVALUATE** doit inclure une **clause WHEN OTHER** pour **gérer les cas non couverts par les clauses précédentes**, assurant ainsi la robustesse du code.

```
EVALUATE TRUE
  WHEN AGE >= 65
    MOVE 'SENIOR' TO CATEGORY
  WHEN AGE >= 18 AND AGE < 65
    MOVE 'ADULT' TO CATEGORY
  WHEN AGE < 18
    MOVE 'MINOR' TO CATEGORY
  WHEN OTHER
    MOVE 'UNSPECIFIED' TO CATEGORY
END-EVALUATE
```

La clause **WHEN OTHER** capture tous les autres cas non prévus par les conditions ci-dessus, par exemple si **AGE** n'est pas correctement défini ou est négatif, et assigne la valeur 'UNSPECIFIED' à **CATEGORY**. Cela assure que la variable CATEGORY reçoit toujours une valeur, **ce qui augmente la robustesse du code en évitant des comportements imprévus ou des erreurs lors de valeurs non attendues**.

Utilisation des Parenthèses pour les conditions

Lors de l'utilisation des opérateurs logiques **AND**, **OR**, ou **NOT** dans les **instructions IF**, **encadrez chaque condition avec des parenthèses pour éviter toute ambiguïté**.

```
IF (AGE >= 18 AND AGE <= 65)
  AND (STATUT-CIVIL = 'CELIBATAIRE'
    OR STATUT-CIVIL = 'DIVORCE')
  MOVE 'ADMISSIBLE' TO STATUT-ASSURANCE
ELSE
  MOVE 'NON-ADMISSIBLE' TO STATUT-ASSURANCE
END-IF
```

- Chaque condition au sein de l'instruction **IF** est clairement isolée sur une nouvelle ligne, facilitant la compréhension de la logique.
- Les parenthèses sont utilisées pour encadrer les groupes de conditions, clarifiant la manière dont les opérateurs **AND** et **OR** sont évalués.
- L'ajout de **END-IF** à la fin assure que la structure de contrôle est correctement fermée, ce qui améliore la maintenance et la lisibilité du code.

Conclusion

Au terme de ce voyage à travers les meilleures pratiques en **COBOL**, vous avez désormais une base solide et des stratégies avancées pour écrire, optimiser et maintenir des programmes **COBOL** de manière efficace. De la compréhension de la syntaxe fondamentale à l'intégration du **COBOL** dans des environnements technologiques modernes, ce guide a pour but de vous équiper des outils nécessaires pour exceller dans le développement **COBOL**.

Ce récapitulatif des meilleures pratiques en **COBOL** n'est que le début de votre parcours d'apprentissage. La maîtrise du **COBOL**, un langage ayant une importance critique dans les infrastructures informatiques mondiales, ouvre la porte à d'innombrables opportunités professionnelles et techniques. Pour ceux désireux d'approfondir leurs compétences et de se plonger encore plus dans le développement **COBOL**, la formation complète **FromZeroToCobol** offre une opportunité unique.

Nous vous invitons à vous inscrire à **FromZeroToCobol** pour une immersion profonde dans le monde du développement **COBOL**.

Conclusion

Que vous soyez un débutant cherchant à comprendre les bases ou un professionnel expérimenté souhaitant rafraîchir et étendre vos compétences, cette formation a été conçue pour répondre à vos besoins. Avec un accent sur l'apprentissage pratique, **FromZeroToCobol** vous guidera à travers des projets réels, vous permettant d'appliquer les connaissances acquises de manière concrète et significative.

N'attendez plus pour prendre votre carrière en main et devenir un expert reconnu en **COBOL**. Rejoignez la formation **FromZeroToCobol** et commencez dès aujourd'hui à construire votre avenir dans le développement **COBOL**. L'innovation et l'excellence vous attendent.

FromZeroToCobol