Exercices

Ces exercices vous donneront des exemples d'algorithmes répondant à une problématiques données. L'ensemble de ces algorithmes peut former une base de révision algorithmique car ce document couvre les éléments nécessaires pour construire la plupart des algorithmes procéduriaux existant.

Problèmes

Vous trouverez la liste des problèmes et une de leurs solutions algorithmiques ci-dessous.

3. Vérifier si un utilisateur est majeur

Problème : Vérifier si un utilisateur est majeur grâce à son âge. Mettre une variable `resultat` à `Vrai` si cela est le cas, sinon mettez la à `Faux`.

```
age <- 25
```

Algorithme :

```
@Si age >= 18
    @DebutBloc
    resultat <- Vrai
    @FinBloc
@Sinon
    @DebutBloc
    resultat <- Faux
    @FinBloc
```

ou

```
resultat <- Faux
@Si age >= 18
    @DebutBloc
    resultat <- Vrai
    @FinBloc
```

En fonction :

```
@Fonction estMajeur @Entrée age
@DebutBloc
    @Si age >= 18
        @DebutBloc
        @Retouner Vrai
        @FinBloc
    @Sinon
        @DebutBloc
        @Retouner Faux
        @FinBloc
@FonBloc
```

ou

```
@Fonction estMajeur @Entrée age
@DebutBloc
    resultat <- Faux
    @Si age >= 18
        @DebutBloc
        resultat <- Vrai
        @FinBloc
    @Retourner resultat
@FonBloc
```

4. Vérifier si un utilisateur est mineur

Problème : Vérifier si un utilisateur est mineur grâce à son âge. Mettre une variable `resultat` à `Vrai` si cela est le cas, sinon mettez la à `Faux`.

```
age <- 12
```

Algorithme :

```
resultat <- Faux
@Si age < 18
    @DebutBloc
    resultat <- Vrai
    @FinBloc
```

En fonction :

```
@Fonction estMineur @Entrée age
@DebutBloc
    resultat <- Faux
    @Si age < 18
        @DebutBloc
        resultat <- Vrai
        @FinBloc
    @Retourner resultat
@FinBloc
```

5. Echanger les 2 premiers éléments d'un tableau

Problème : Echanger les 2 premiers éléments d'un tableau

```
tab <- [23, 4, 2, 543, 34, ...]
```

Algorithme :

```
A <- tab[0]
tab[0] <- tab[1]
tab[1] <- A
```

En fonction :

```
@Fonction echanger @Entrée tab
@DebutBloc
    A <- tab[0]
    tab[0] <- tab[1]
    tab[1] <- A
    @Retourner tab
@FinBloc
```

6. Compter le nombre d'éléments dans un tableau

Problème : Compter le nombre d'éléments dans un tableau. Mettre le compte dans `resultat`.

```
tab <- [23, 4, 2, 543, 34, ...]
```

Algorithme :

```
resultat <- 0
@PourChaque element @Dans tab
    @DebutBloc
    resultat <- resultat + 1
    @FinBloc
```

En fonction :

```
@Fonction compter @Entrée tab
@DebutBloc
    resultat <- 0
    @PourChaque element @Dans tab
        @DebutBloc
        resultat <- resultat + 1
        @FinBloc
    @Retourner resultat
@FinBloc
```

7. Faire la somme des éléments d'un tableau

Problème : Faire la somme des éléments d'un tableau

```
tab <- [23, 4, 2, 543, 34, ...]
```

Algorithme :

```
resultat <- 0
@PourChaque prix_en_cours @Dans tab
    @DebutBloc
    resultat <- resultat + prix_en_cours
    @FinBloc
```

ou

```
compte <- 0
@PourChaque element @Dans tab
    @DebutBloc
    compte <- compte + 1
    @FinBloc

resultat <- 0
@Pour i @De 0 @A compte -1
    @DebutBloc
    resultat <- resultat + tab[i]
    @FinBloc
```

En fonction :

```
@Fonction sommer @Entrée tab
@DebutBloc
    resultat <- 0
    @PourChaque prix_en_cours @Dans tab
        @DebutBloc
        resultat <- resultat + prix_en_cours
        @FinBloc
    @Retourner resultat
@FinBloc
```

ou

```
@Fonction sommer @Entrée tab
@DebutBloc
    compte <- 0
    @PourChaque element @Dans tab
        @DebutBloc
        compte <- compte + 1
        @FinBloc

    resultat <- 0
    @Pour i @De 0 @A compte -1
        @DebutBloc
        resultat <- resultat + tab[i]
        @FinBloc

    @Retourner resultat
@FinBloc
```

8. Trouver l'élément maximum d'un tableau

Problème : Trouver l'élément maximum d'un tableau comprenant des nombres de 0 à 1000. Mettre le numbre maximum dans `resultat`.

```
tab <- [...]
```

Algorithme :

```
resultat <- 0
@PourChaque prix_en_cours @Dans tab
    @Debutbloc
    @Si prix_en_cours > resultat
        @DebutBloc
        resultat <- prix_en_cours
        @FinBloc
    @FinBloc
```

En fonction :

```
@Fonction trouverMaximum @Entrée tab
@DebutBloc
    resultat <- 0
    @PourChaque prix_en_cours @Dans tab
        @Debutbloc
        @Si prix_en_cours > resultat
            @DebutBloc
            resultat <- prix_en_cours
            @FinBloc
        @FinBloc
    @Retourner resultat
@FinBloc
```

9. Trouver l'élément minimum d'un tableau

Problème : Trouver l'élément minimum d'un tableau comprenant des nombres de 0 à 1000

```
tab <- [...]
```

Algorithme :

```
resultat <- tab[0]
@PourChaque prix_en_cours @Dans tab
    @Debutbloc
    @Si prix_en_cours < resultat
        @DebutBloc
        resultat <- prix_en_cours
        @FinBloc
    @FinBloc
```

En fonction :

```
@Fonction trouverMinimum @Entrée tab
@DebutBloc
    resultat <- tab[0]
    @PourChaque prix_en_cours @Dans tab
        @Debutbloc
        @Si prix_en_cours < resultat
            @DebutBloc
            resultat <- prix_en_cours
            @FinBloc
        @FinBloc
    @Retourner resultat
@FinBloc
```

10. Trouver le premier élément supérieur à 500

Problème : Trouver le premier élément supérieur à 500. S'il n'y en a pas, le resultat doit être 0

```
tab <- [23, 4, 2, 543, 34, ...]
```

Algorithme :

```
resultat <- 0
@PourChaque element @Dans tab
    @DebutBloc
    @Si element > 500
        @DebutBloc
        resultat <- element
        @Stop
        @FinBloc
    @FinBloc
```

ou

```
longueur <- 0
@PourChaque element @Dans tab
    @DebutBloc
    longueur <- longueur + 1
    @FinBloc

i <- 0
@TantQue tab[i] < 500 @Et i <= longueur - 1
    @DebutBloc
    i <- i + 1
    @FinBloc

resultat <- 0
@Si tab[i] >= 500
@DebutBloc
    resultat <- tab[i]
@FinBloc
```

En fonction :

```
@Fonction trouverPremierSup500 @Entrée tab
@DebutBloc
    resultat <- 0
    @PourChaque element @Dans tab
        @DebutBloc
        @Si element > 500
            @DebutBloc
            resultat <- element
            @Stop
            @FinBloc
        @FinBloc
    @Retourner resultat
@FinBloc
```

ou plus rapide

```
@Fonction trouverPremierSup500 @Entrée tab
@DebutBloc
    @PourChaque element @Dans tab
        @DebutBloc
        @Si element > 500
            @DebutBloc
            @Retourner element
            @FinBloc
        @FinBloc
@FinBloc
```

ou

```
@Fonction trouverPremierSup500 @Entrée tab
@DebutBloc
    longueur <- 0
    @PourChaque element @Dans tab
        @DebutBloc
        longueur <- longueur + 1
        @FinBloc

    i <- 0
    @TantQue tab[i] < 500 @Et i <= longueur - 1
        @DebutBloc
        i <- i + 1
        @FinBloc

    resultat <- 0
    @Si tab[i] >= 500
    @DebutBloc
        resultat <- tab[i]
    @FinBloc

    @Retourner resultat
@FinBloc
```

11. Copier un tableau

Problème : Copier le tableau `tab` dans un autre tableau `autre`

```
tab <- [23, 4, 2, 543, 34, ...]
```

Algorithme :

```
autre <- []
@PourChaque element @Dans tab
    @DebutBloc
    Ajouter element Dans autre
    @FinBloc
```

En fonction :

```
@Fonction copierTableau @Entrée tab
@DebutBloc
    autre <- []
    @PourChaque element @Dans tab
        @DebutBloc
        Ajouter element Dans autre
        @FinBloc
    @Retourner autre
@FinBloc
```

12. Copier les premiers éléments d'un tableau dont la somme fait au moins 500

Problème : Copier les premiers éléments d'un tableau dont la somme fait au moins 500

```
tab <- [23, 4, 2, 543, 34, ...]
```

Algorithme :

```
autre_tableau <- []
somme <- 0
@PourChaque prix_en_cours @Dans tab
    @DebutBloc
    somme <- somme + prix_en_cours
    Ajouter prix_en_cours Dans autre_tableau
    @Si somme >= 500
        @DebutBloc
        @Stop
        @FinBloc
    @FinBloc
```

En fonction :

```
@Fonction copierElementPourSomme500 @Entrée tab
@DebutBloc
    autre_tableau <- []
    somme <- 0
    @PourChaque prix_en_cours @Dans tab
        @DebutBloc
        somme <- somme + prix_en_cours
        Ajouter prix_en_cours Dans autre_tableau
        @Si somme >= 500
            @DebutBloc
            @Stop
            @FinBloc
        @FinBloc
    @Retouner autre_tableau
@FinBloc
```