

Exercice 1 : Requêtes algorithmiques

Sans charger de données, formuler dans BaseX les requêtes suivantes en utilisant des expressions FLWOR.

- 1 Dans une clause `let`, initialiser une variable avec une chaîne de caractères quelconque (ex. « Hello World! ») et afficher son contenu dans un élément XML `<resultat> </resultat>` spécifié dans la clause `return`.
- 2 Dans une nouvelle requête, initialiser deux variables avec des valeurs numériques et calculer leur somme. Résultat attendu :

```
<resultat>
  <a val="3" />
  <b val="2" />
  <somme val="5" />
</resultat>
```
- 3 Afficher la somme de tous les entiers de 1 à 10 dans un élément XML `<somme> </somme>`. Résultat attendu : 55. Indication : utiliser une clause `let` et la fonction d'agrégation `sum()`.
- 4 Afficher la table de multiplication de $i \times j$ avec $i, j = 1..10$. Chaque ligne de la table de multiplication doit être formatée comme suit : `<resultat><i>2</i><j>3</j><p>6</p></resultat>` (ici, $i = 2$ et $j = 3$). Indication : utiliser une clause `for`.

Exercice 2 : Requêtes sur données

Télécharger le document `nutrition.xml` dont l'adresse est donnée ci-dessous et l'importer dans une base de données BaseX.

<https://eric.univ-lyon2.fr/jdarmont/docs/nutrition.xml>

Formuler ensuite les requêtes suivantes à l'aide d'expressions FLWOR.

- 1 Toutes les valeurs journalières (*daily values*).
- 2 Noms de tous les aliments (*food*).
- 3 Même question en triant le résultat par ordre alphabétique inverse.
- 4 Noms de tous les aliments triés par ordre croissant de total de calories. Indiquer les calories en attribut du résultat pour vérifier s'il est correct. Conclusion ? Corriger le problème !
- 5 Noms de tous les aliments triés par ordre décroissant de total de gras (*fat*) et par ordre croissant de gras saturé.
- 6 Noms et fabricants (*mfr*) de tous les aliments.
- 7 Noms et positions dans le document `nutrition.xml` de tous les aliments, au format `<foodstuff pos=""> </foodstuff>`.
- 8 Noms des aliments servis en portion (*serving*) supérieure à 100
- 9 Même question avec les unités indiquées en attribut.
- 10 Noms des aliments dont le total de calories et de gras est supérieur à 100.
- 11 Aliments (toutes les caractéristiques) dont le total de gras dépasse 10 % de la valeur journalière.
- 12 Aliments (toutes les caractéristiques) pour lesquels n'importe quelle valeur (*total fat, saturated fat, cholesterol, sodium, carbonate, fiber* ou *protein*) dépasse 10 % de sa valeur journalière.

- 13 Noms des aliments dont le taux de vitamine C est supérieure ou égale à 10 entre les balises `<high-in-vitaminC>` `</high-in-vitaminC>`, tandis que les autres aliments sont étiquetés `<low-in-vitaminC>` `</low-in-vitaminC>`. Indiquer le taux de vitamine C en attribut pour vérification.
- 14 Nombre d'aliments.
- 15 Valeurs moyennes des caractéristiques des aliments, de *serving* à fer (*fe*).
- 16 Même question, mais inclure les unités comme attributs lorsque c'est possible. Faire la moyenne des quantités servies a-t-il un sens ?

Correction

(: Exercise 1 :)

```
(: 1 :)
let $x := "Hello World!"
return <resultat>{$x}</resultat>
```

```
(: 2.1 :)
let $a := 3,
    $b := 2
return <resultat>
  <a val="{ $a }" />
  <b val="{ $b }" />
  <somme val="{ $a + $b }" />
</resultat>
```

```
(: 2.2 séparation calcul XQuery-retour XML:)
let $a := 3,
    $b := 2,
    $s := $a + $b
return <resultat>
  <a val="{ $a }" />
  <b val="{ $b }" />
  <somme val="{ $s }" />
</resultat>
```

```
(: 3 :)
let $s := sum(1 to 10)
return <somme>{$s}</somme>
```

```
(: 4 :)
for $i in 1 to 10,
    $j in 1 to 10
let $p := $i * $j
return <resultat><i>{$i}</i> <j>{$j}</j> <p>{$p}</p></resultat>
```

(: Exercise 2 :)

```
(: 1 :)
for $v in //daily-values/*
return $v
```

```
(: 2 :)
for $n in //name
return $n
```

```
(: 3 :)
for $n in //name
order by $n descending
return $n
```

```
(: 4 :)
for $f in //food
order by number($f/calories/@total)
return
  <foodstuff total-calories="{data($f/calories/@total)}">
    {data($f/name)}
  </foodstuff>
```

```

(: 5 :)
for $f in //food
order by number($f/total-fat) descending, number($f/saturated-fat)
return <foodstuff total-fat="{data($f/total-fat)}"
        saturated-fat="{data($f/saturated-fat)}">
        {data($f/name)}
</foodstuff>

(: 6 :)
for $f in //food
return <food>
        {$f/name}
        {$f/mfr}
</food>

(: 7 :)
for $f at $i in //food/name
return <foodstuff pos="{ $i }">{data($f)}</foodstuff>

(: 8 :)
for $f in //food
where $f/serving > 100
return <foodstuff>{data($f/name)}</foodstuff>

(: 9 :)
for $f in //food
where $f/serving > 100
return <foodstuff units="{data($f/serving/@units)}">{data($f/name)}</foodstuff>

(: 10 :)
for $f in //food
where $f/calories/@total > 100 and $f/calories/@fat > 100
return <foodstuff>{data($f/name)}</foodstuff>

(: 11 :)
for $f in //food
where $f/total-fat > //daily-values/total-fat / 10
return $f

(: 12 :)
for $f in //food
where $f/total-fat > //daily-values/total-fat / 10
or $f/saturated-fat > //daily-values/saturated-fat / 10
or $f/cholesterol > //daily-values/cholesterol / 10
or $f/sodium > //daily-values/sodium / 10
or $f/carb > //daily-values/carb / 10
or $f/fiber > //daily-values/fiber / 10
or $f/protein > //daily-values/protein / 10
return $f

(: 12bis - Alexy Lafosse, 2024 :)
for $f in //food
where (some $v in //daily-values/*
        satisfies number(data($f/*[name() = name($v)])) > number(data($v)) * 0.1)
return $f
(: lisibilité vs. concision :)

```

```

(: 13 :)
for $f in //food
return   if ($f/vitamins/c >= 10)
         then <high-in-vitaminC rate="{data($f/vitamins/c)}">
             {data($f/name)}
             </high-in-vitaminC>
         else <low-in-vitaminC rate="{data($f/vitamins/c)}">
             {data($f/name)}
             </low-in-vitaminC>

(: 14 :)
let $c := count(//food)
return <foodstuff-count>{$c}</foodstuff-count>

(: 15 & 16 :)
let $avg-serving := avg(//food/serving),
    $serving-units := distinct-values(//serving/@units),
    $avg-total-calories := avg(//food/calories/@total),
    $avg-fat-calories := avg(//food/calories/@fat),
    $avg-total-fat := avg(//food/total-fat),
    $avg-saturated-fat := avg(//food/saturated-fat),
    $avg-cholesterol := avg(//food/cholesterol),
    $avg-sodium := avg(//food/sodium),
    $avg-carb := avg(//food/carb),
    $avg-fiber := avg(//food/fiber),
    $avg-protein := avg(//food/protein),
    $avg-vitaminA := avg(//food/vitamins/a),
    $avg-vitaminC := avg(//food/vitamins/c),
    $avg-ca := avg(//food/minerals/ca),
    $avg-fe := avg(//food/minerals/fe)
return   <averages>
         <serving units="{ $serving-units}">{$avg-serving}</serving>
         <total-calories>{$avg-total-calories}</total-calories>
         <fat-calories>{$avg-fat-calories}</fat-calories>
         <total-fat units="{data(//daily-values/total-fat/@units)}">
             { $avg-total-fat }
         </total-fat>
         <saturated-fat units="{data(//daily-values/saturated-fat/@units)}">
             { $avg-saturated-fat }
         </saturated-fat>
         <cholesterol units="{data(//daily-values/cholesterol/@units)}">
             { $avg-cholesterol }
         </cholesterol>
         <sodium units="{data(//daily-values/sodium/@units)}">
             { $avg-sodium }
         </sodium>
         <carb units="{data(//daily-values/carb/@units)}">
             { $avg-carb }
         </carb>
         <fiber units="{data(//daily-values/fiber/@units)}">
             { $avg-fiber }
         </fiber>
         <protein units="{data(//daily-values/protein/@units)}">
             { $avg-protein }
         </protein>
         <vitaminA>{$avg-vitaminA}</vitaminA>
         <vitaminC>{$avg-vitaminC}</vitaminC>
         <ca>{$avg-ca}</ca>
         <fe>{$avg-fe}</fe>
         </averages>

```