

Les blocs et les enumerateurs

Main Author(s): to be fixed: B. Pottier, Université de Brest, Bernard.Pottier@univ-brest.fr

Currently developed on: VisualWorks, Can be used for lectures using ?Squeak/VisualWorks?, latest version ?1.2? available on Université de Brest —B. Pottier, Université de Brest, Bernard.Pottier@univ-brest.fr al. , contact person: ??

1.1 TP

On affiche une chaîne dans le Transcript en lui envoyant le message `show: uneChaine`. On passe à la ligne dans le Transcript en lui envoyant le message `cr`

Exemple : Transcript show: 'Salut'. Transcript cr.

en utilisant la cascade (plusieurs messages envoyés au même objet)

Transcript show: 'Salut'; cr.

1.1.1 Les blocs

L'évaluation d'un bloc sans paramètre s'obtient en envoyant le message `value` au bloc. Les instructions du bloc sont exécutées et le résultat de la dernière instruction est retourné.

Un bloc avec une variable s'évalue en envoyant le message `value: uneValeur`.

Un bloc avec deux variables s'évalue en envoyant le message

`value: unePremiereValeur value: uneDeuxiemeValeur` (ceci jusqu'à 4 variables).

1. Ecrire un bloc (sans variables) qui calcule la racine carrée de 1, puis de 2, puis de 3. Evaluer ce bloc.
2. Ecrire un bloc avec une variable, qui renvoie le carré de cette variable. Evaluer ce bloc.
3. Ecrire un bloc avec deux variables, qui renvoie la plus grande de ces deux variables (utiliser la méthode `max:`). Evaluer ce bloc.

1.1.2 Les méthodes d'intervalle

Les nombres comprennent les messages suivants : `to: uneValeurArret do: unBloc`

et : `to: uneValeurArret by: unPas do: unBloc`

Exemple qui affiche les nombres de 1 à 10 dans le Transcript

1 to: 10 do: [:i | Transcript show: i printString]

Exemple qui affiche, par pas de 2, les nombres de 1 à 10 dans le Transcript

1 to: 10 by: 2 do: [:i | Transcript show: i printString]

1. Convertir le nombre 65 (représentant un code ASCII) en Character (`asCharacter`), puis en Symbol (`asSymbol`), puis en String (`asString`).
2. Afficher dans le Transcript les caractères compris entre les codes ASCII 65 et 122 (bornes incluses).
3. Afficher les nombres impairs de 1 à 33 dans le Transcript

1.1.3 Les énumérateurs

Toutes les sous-classes de **Collection** comprennent ces messages, appelés *énumérateurs* :

- **do**: unBloc évalue unBloc sur chaque élément de la collection,
- **collect**: unBloc comme **do**: mais renvoie une collection des résultats,
- **select**: unBloc évalue unBloc sur chaque élément et renvoie ceux pour qui l'évaluation renvoie **true**,
- **reject**: unBloc évalue unBloc sur chaque élément et renvoie ceux pour qui l'évaluation renvoie **false**,
- **detect**: unBloc renvoie le premier élément pour qui l'évaluation renvoie **true**,
- **detect: unBloc ifNone: unAutreBloc** a le même comportement que **detect**: mais permet d'exécuter le deuxième bloc (**unAutreBloc**) s'il n'y a pas d'élément pour qui l'évaluation de **unBloc** renvoie **true**.
- **inject: uneValeur into: unBlocBinaire** injecte le résultat de l'exécution précédente de **unBlocBinaire** (un bloc à deux paramètres) dans la suivante.
uneCollection inject: valeur into: [arg1 arg2 ...]
arg1 est initialisé avec **valeur**,
arg2 prend successivement la valeur de chaque élément et évalue le bloc avec cette valeur (comme un **do**:),
à l'issue de l'évaluation courante, le résultat de l'évaluation est affectée dans **arg1**.

L'énumérateur **do**:

1. Faire la somme des éléments d'un tableau.
2. La méthode **constantNames** envoyée à la classe **ColorValue** renvoie un tableau **constant** de symboles, chaque symbole ayant le nom d'une couleur.
Afficher, dans le Transcript, les couleurs **constantes** de la classe **ColorValue**

L'énumérateur **collect**:

1. Construire un premier tableau, **et avec la méthode collect**:, construire un deuxième tableau identique au premier.
2. A partir d'un premier tableau, construire un deuxième tableau dont la valeur de chaque élément est le double de l'élément correspondant dans le premier tableau.

Autres énumérateurs

1. A partir du tableau **constant** de symboles ayant le nom d'une couleur, détecter la première couleur commençant par un **d**
(on obtient le premier caractère d'un symbole avec **first**).
2. A partir du tableau **constant** de symboles ayant le nom d'une couleur, construire un tableau des noms de couleur commençant par un **d**
3. A partir du tableau **constant** de symboles ayant le nom d'une couleur, construire un tableau des noms de couleur ne commençant pas par un **d**
Afficher (avec un **do**:) ce tableau dans le Transcript

1.1.4 Les structures alternatives

Smalltalk définit sur la classe **True** et sur la classe **False** quatre méthodes d'instance `ifTrue:`, `ifFalse:`, `ifTrue:ifFalse:`, `ifFalse:ifTrue:` ¹.

	Classe True	Classe False
Méthode: Action:	<code>ifTrue: unBloc</code> renvoyer l'évaluation de <code>unBloc</code>	<code>ifTrue: unBloc</code> renvoyer nil
Méthode: Action:	<code>ifFalse: unBloc</code> renvoyer nil	<code>ifFalse: unBloc</code> renvoyer l'évaluation de <code>unBloc</code>
Méthode: Action:	<code>ifTrue: unBloc ifFalse: unAutreBloc</code> renvoyer l'évaluation de <code>unBloc</code>	<code>ifTrue: unBloc ifFalse: unAutreBloc</code> renvoyer l'évaluation de <code>unAutreBloc</code>
Méthode: Action:	<code>ifFalse: unBloc ifTrue: unAutreBloc</code> renvoyer l'évaluation de <code>unAutreBloc</code>	<code>ifFalse: unBloc ifTrue: unAutreBloc</code> renvoyer l'évaluation de <code>unBloc</code>

Par ailleurs, les opérateurs logiques **&** (conjonction, le ET), **Eqv** (équivalence), **not** (négation), **xor** (ou exclusif), **|** (disjonction, le OU) sont définies sur les classes **True** et **False**.

1. Tester si un nombre est impair (en lui envoyant le message `odd`) et sonner la cloche (`Screen default ringBell`) si c'est vrai.
2. Sans utiliser `max:`, écrire un bloc avec deux paramètres qui renvoie le maximum des deux paramètres

1.1.5 Enumérateurs et alternatives

1. Faire la somme des éléments positifs d'un tableau
2. Créer un tableau de 0 ou 1 à partir d'un tableau existant, un nombre du tableau existant est remplacé par un 0 s'il est supérieur ou égal à 10, par un 1 s'il est inférieur à 10.

1.1.6 Itération de blocs

Utilisation des messages `timesRepeat:`, `repeat`, `whileTrue:`

1. Ecrire 10 fois la chaîne 'coucou' dans le Transcript (en passant à la ligne après chaque 'coucou').
2. Ecrire un bloc avec une variable `n` qui écrit `n` fois la chaîne 'coucou' dans le Transcript (en passant à la ligne après chaque 'coucou'). Evaluer ce bloc.
3. Itérer avec un `repeat` un bloc qui incrémente un compteur de 1. On s'arrête quand le compteur est supérieur à 10.
4. Créer un objet de la classe `Time` à 3 secondes du temps courant.
`Time now addTime: (Time fromSeconds: 3)`
Boucler jusqu'à ce que le temps courant dépasse cet objet, en passant à la ligne dans le Transcript

1.2 Exercices (TD)

1.2.1 Les blocs

1. Ecrire un bloc avec une variable, qui renvoie la conversion en degrés Celsius de cette variable (supposée être en Fahrenheit). Evaluer ce bloc ($C = (5/9) (F - 32)$).
2. Ecrire un bloc avec deux variables, qui convertit ces variables en `String`, les concatène en les séparant avec un blanc et renvoie le résultat de la concaténation.

¹Un truc : Si on tape `< Control > t` (ou `< Control > f`) dans une fenêtre de code, le système insère `ifTrue: (ifFalse:)`

1.2.2 Les énumérateurs

L'énumérateur **do**:

1. Compter le nombre d'éléments d'un tableau. Vérifier avec **size**
2. Faire la moyenne des éléments d'un tableau.

L'énumérateur **collect**:

1. Une chaîne instance de la classe **cString** est un tableau, donc on peut avoir son premier élément (**first**), son *i*-ème élément (**at: i**), affecter une **Valeur** dans son *i*-ème élément **at: i put: uneValeur**, etc
A partir du tableau **constant** de symboles ayant le nom d'une couleur, construire un tableau de **String** où le nom de la couleur commence par une majuscule (**asUppercase**).
2. Construire un tableau dont la valeur des éléments est la somme de l'élément précédent avec l'élément courant.

1.2.3 Les structures alternatives

1. Sans utiliser **max:**, écrire un bloc avec trois paramètres qui renvoie le maximum des trois paramètres
2. Ecrire un bloc avec un paramètre, qui teste si le paramètre est une minuscule (**isLowercase**), une majuscule (**isUppercase**), un chiffre (**isDigit**) ou autre.
Le bloc renvoie 'minuscule' ou 'majuscule' ou 'chiffre' ou 'autre'.

1.2.4 Les intervalles

Méthodes d'intervalles

1. Afficher les 10 premiers carrés.
2. Afficher les multiples de 10.

Créer et utiliser des intervalles

Envoyé à un nombre, la méthode **to: borneSuperieure** renvoie un **Interval** allant du nombre à la **borneSuperieure** par pas de 1.

Envoyé à un nombre, la méthode **to: borneSuperieure by: lePas** renvoie un **Interval** allant du nombre à la **borneSuperieure** par pas de **lePas**.

Les **Interval** étant des collections, on peut utiliser les énumérateurs **do:**, **collect:**, etc.

Exemple : **tab := (1 to: 100)** crée un tableau des 100 premiers entiers.

1. Calculer la somme des 100 premiers nombres entiers à l'aide d'un tableau. Vérifier avec la formule $n*(n+1)/2$
2. Créer un tableau avec les nombres de 0 à 360 de 30 en 30. utiliser ce tableau pour afficher la table des sinus de 30 degrés en 30 degrés.

1.2.5 Enumérateurs et alternatives

1. A partir d'un premier tableau, construire un tableau en inversant tous les éléments qui sont des fractions.

1.2.6 Autres énumérateurs

1. A partir d'un premier tableau, construire un deuxième tableau en supprimant les éléments négatifs.
2. Définir la somme des éléments d'un tableau avec `inject:into:`.
3. Définir la somme des éléments positifs d'un tableau avec `select:` puis `inject:into:`.
4. Définir la somme des éléments négatifs d'un tableau avec `reject:` puis `inject:into:`.

1.2.7 Itération de blocs

1. Afficher 10 * dans le Transcript, puis passe à la ligne.
2. Afficher 10 lignes de 10 * dans le Transcript (en passant à la ligne après chaque ligne de 10 *).
3. Ecrire un bloc à deux paramètres `ctl` et `c`, qui affiche `l` lignes de `c` colonnes de * dans le Transcript (en passant à la ligne après chaque ligne).