Valeur, variable, référence

Table des matières

Introduction

Nous savons utiliser des variables, leur définir un type et une valeur, ainsi que les passer en paramètres à des fonctions. Dans ce module nous allons aborder différentes notions plus avancées avec les variables, comme par exemple la différence entre la copie et la référence à une variable. Nous apprendrons aussi que le type d'une variable n'est pas forcément figé, que cela dépend des langages, mais qu'il est possible de le changer.

Enfin nous étudierons le mécanisme de passage d'une variable à une fonction, qui, selon comment il est réalisé, peut donner des résultats différents.

I Valeur, variable, référence

Objectifs

- Connaître et comprendre les notions de valeur, variable et référence ;
- Connaître la notion de type primitif et de type composé.

Mise en situation

Une variable possède une valeur, qui est stockée en mémoire, et qu'il est possible de copier. Mais il est possible de réaliser une autre opération, la référence.

Introduction par l'exemple

Exemple

On se donne un exemple simple en Python et JavaScript :

```
1 solution = 42
```

Ici, on dit couramment que « la variable `solution` a pour valeur 42 ». Mais qu'est-ce qu'une valeur ? Et qu'est-ce qu'une variable ?

Valeur

Az Définition

Une valeur est une information constante stockée sous la forme d'une séquence de bits et qui dispose d'un type.

Reprise de l'exemple

Exemple

Dans l'exemple donné plus haut :

```
1 42
```

... est la valeur utilisée. Elle est de type **entier**.

Il peut y en avoir d'autres valeurs, comme celle de type chaînes de caractères :

```
1 "Ceci est une autre valeur"
```

Ces valeurs sont stockées en mémoire mais ne sont pas **référencées**.

Variable

Az Définition

Une variable est un symbole qui **référence** une valeur stockée en mémoire.

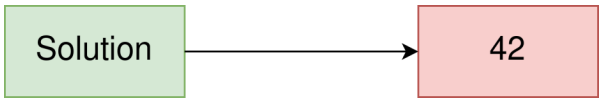
Reprise de l'exemple

Exemple

Dans l'exemple donné plus haut :

```
solution
```

... est la variable associée à la valeur 42.

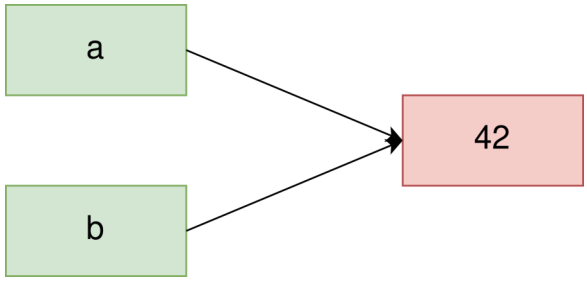


Référencement d'une valeur par une variable

On peut tout à fait définir plusieurs variables pour une même valeur :

```
a = 42
b = 42
```

Ici a et b sont deux variables **référençant** la même valeur, 42.



Référence d'une même valeur par deux variables

Affectation et stockage des valeurs

Fondamental

Lorsque l'on associe une variable à une valeur, on utilise une **affectation** avec l'opérateur =.

```
a = 42
```

Dans cet exemple, la valeur de celle-ci est directement stockée en mémoire à une adresse particulière.

Stockage de valeur et référencement

Fondamental

Lorsque l'on utilise une affectation, les valeurs sont stockées en mémoire à une adresse spécifique.

La variable stocke ensuite une **référence** vers l'adresse de cette valeur. En d'autres termes, **une variable peut changer de valeur.**

À retenir

Lorsque l'on travaille avec un langage de programmation, on utilise des variables qui permettent de manipuler des valeurs via un système de référence.

Ces valeurs ont un type et les variables peuvent changer de valeurs.

Exercice : Appliquer la notion

[solution n°1 p. 31]

On se donne le morceau de code suivant :

```
carNumbers = 1254358
moto = 'We do what we can because we must'
quote = moto
console.log('18126798')
console.log(24)
```

Associer les mots aux différentes cibles.

A `'We do what we can because we must'` **B** `carNumbers` **C** `quote` **D** `moto`

E `1254358` **F** `24` **G** `'18126798'`

Variable	Valeur (entier)	Valeur (chaîne de caractères)

III Typage dynamique et affectation

Objectifs

- Connaître la notion de typage dynamique ;
- Connaître le comportement de l'assignation pour Python et JavaScript.

Mise en situation

On peut distinguer différentes manières de typer les variables selon les langages. Certains langages ont un typage statique, d'autre un typage dynamique. Cela signifie que les premiers ne permettent pas de changer le type d'une variable après sa déclaration. C'est souvent le cas des langages de bas niveau, qui sont proches de la machine et de sa gestion mémoire. Les langages à typage dynamique sont eux beaucoup plus flexibles, ils permettent de changer le type d'une variable tout au long de l'exécution.

JavaScript et Python : des langages au typage dynamique Rappel

JavaScript et Python sont des langages au typage dynamique : les variables peuvent changer de valeurs même si celles-ci ont un type différent. Dans ce cadre, les types sont portés par les **valeurs** et non les variables.

Typage dynamique Exemple

Ici, a est une variable qui change de type pour passer d'un entier à une chaîne de caractères.

```
1 a = 42
2 a = 'fezfe'
```

Type primitif Rappel

Les types primitifs sont les types de valeurs les plus simples qui ne contiennent qu'une information atomique.

Types primitifs en Python et JavaScript Exemple

Il y a quatre types primitifs communs à Python et JavaScript :

- les entiers,
- les flottants,
- les chaînes de caractères,
- les booléens.

Complément

En JavaScript il y a trois autres types primitifs :

- les BigInt,
- les symboles,
- le type `undefined`.

Type composé

Rappel

Les types composés sont des types construits sur un ensemble de types primitifs, comme les listes, les tableaux ou les objets.

Types composés en JavaScript

Exemple

En JavaScript par exemple, deux exemples de types composés sont :

- les tableaux : [ ]
- les objets (enregistrements) : { }

Comportement de l'assignation pour le typage dynamique

Fondamental

Lorsque l'on assigne une variable en utilisant une autre dans des langages au typage dynamique comme (Python, ou JavaScript), on ne copie pas la valeur : on copie une **référence** vers cette valeur.

Cette subtilité a des implications très fortes lorsque l'on traite avec des types composés.

Tester le référencement d'une valeur par deux variables en JavaScript

Syntaxe

En JavaScript, l'opérateur d'égalité permet de vérifier que deux variables référencent la même valeur ou non.

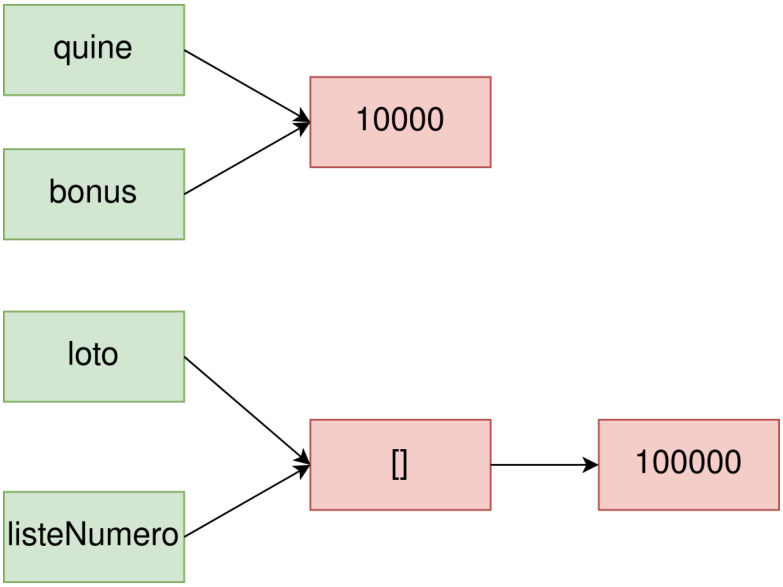
Exemple

```
let quine = 10000
let bonus = quine
console.log('Equalité:', quine === bonus)

let loto = [100000]
let listeNumero = loto
console.log('Equalité:', loto === listeNumero)
```

On obtient :

```
Equalité: true
Equalité: true
```



Conservation de la référence lors de l'affectation d'une variable à une autre

Tester le référencement d'une valeur par deux variables en Python

Syntaxe

En Python la fonction `id` retourne l'identité de l'objet Python stockant la valeur sous-jacente.

Ainsi, avec l'opérateur d'égalité sur ces identités, on peut vérifier que deux variables référencent la même valeur ou non.

Affectation en Python

Exemple

```
1 quine = 10000
2 bonus = quine
3 print('ID de quine:', id(quine))
4 print('ID de bonus:', id(bonus))
5 print('Equalité:', id(a) == id(b))
6
7 loto = [100000]
8 listeNumero = loto
9 print('ID de loto:', id(loto))
10 print('ID de listeNumero:', id(listeNumero))
11 print('Equalité:', id(loto) == id(listeNumero))
```

On obtient :

```
1 ID de quine: 140510876310096
2 ID de bonus: 140510876310096
3 Equalité: True
4 ID de loto: 140080656991808
5 ID de listeNumero: 140080656991808
6 Equalité: True
```

La subtilité est que `quine` et `bonus` ne contiennent pas tous les deux une **copie** de la valeur 10000, mais ils référencent **la même case mémoire**, qui contient la valeur 10000.

Comportement de la modification de valeur

Fondamental

Lorsque l'on modifie une valeur en JavaScript ou en Python à partir d'une variable :

- S'il s'agit d'un type primitif, la valeur est **copiée** puis modifiée. En d'autres termes, **la référence change**.
- S'il s'agit d'un type composé, on ne change pas de référence : seules les références de valeurs du type composés sont modifiées, s'il s'agit de types primitifs. Sinon, la même règle s'applique, et ainsi de suite.

Modification d'un entier en Python

Exemple

En exécutant cela :

```python
quine = 10000
bonus = quine
print('ID de quine:', id(quine))
print('ID de bonus:', id(bonus))
print('Equalité:', id(quine) == id(bonus))

bonus = bonus + 1
print('ID de quine:', id(quine))
print('ID de bonus:', id(bonus))
print('Equalité:', id(quine) == id(bonus))
```

On obtient :

```
ID de quine: 140713190459984
ID de bonus: 140713190459984
Equalité: True
ID de quine: 140713190459984
ID de bonus: 140713190459888
Equalité: False
```

Ce qui montre que dans le cas de modification d'une valeur de type primitif, l'identité (la référence) change.

Modification d'un entier en JavaScript

Exemple

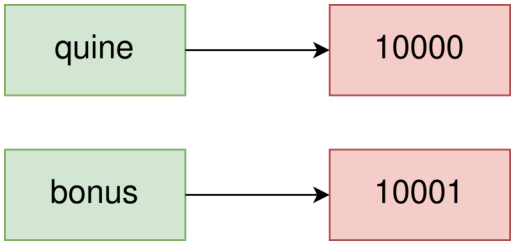
Le comportement est similaire en JavaScript :

```javascript
let quine = 10000
let bonus = quine
console.log('Equalité:', quine === bonus)

bonus = bonus + 1
console.log('Equalité:', quine === bonus)
```

On obtient :

```
Equalité: true
Equalité: false
```



Copie de la valeur référencée lors de la modification d'un type primitif

Modification d'un tableau en Python

Exemple

En exécutant :

```
1 loto = [100000]
2 listeNumero = loto
3 print('ID de loto:', id(loto))
4 print('ID de listeNumero:', id(listeNumero))
5 print('Equalité:', id(loto) == id(listeNumero))
6
7 listeNumero.append(4240142)
8 print('ID de loto:', id(loto))
9 print('ID de listeNumero:', id(listeNumero))
10 print('Equalité:', id(loto) == id(listeNumero))
```

On obtient :

```
1 ID de loto: 140713190902976
2 ID de listeNumero: 140713190902976
3 Equalité: True
4 ID de loto: 140713190902976
5 ID de listeNumero: 140713190902976
6 Equalité: True
```

Ce qui montre que dans le cas de modification d'une valeur de type composé, la référence ne change pas.

Notez que le comportement peut surprendre, car le contenu `loto` est modifié au même titre que le contenu `listeNumero`, car les variables référencent le même contenu : aucune copie n'est effectuée.

Modification d'un tableau en JavaScript

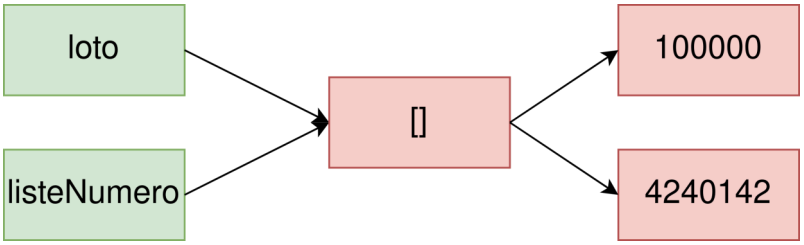
Exemple

Le résultat est similaire en JavaScript :

```
1 let loto = [100000]
2 let listeNumero = loto
3 console.log('Equalité:', loto === listeNumero)
4
5 listeNumero.push(4240142)
6 console.log('Equalité:', loto === listeNumero)
```

On obtient :

```
1 Equalité: true
2 Equalité: true
```



Conservation de la référence lors de la modification d'une valeur de type composé

À retenir

Lors d'une affectation, on assigne des **références** et non des valeurs aux variables. Si on modifie une valeur à travers une variable, seules les références des variables de types primitifs sont modifiées.

Il ne suffit donc pas de faire une affectation pour copier un type composé : la variable originale et la nouvelle variable référenceront la même valeur (tableau, enregistrement, etc.), et les modifications seront reflétées pour les deux variables.

IV Exercice : Appliquer la notion

Question 1

[solution n°2 p. 31]

On se donne le script suivant :

```
1 prixAbricot = 4
2 prixPeche = 4
```

Modifier le code pour savoir si ces deux variables référencent la même valeur ?

Question 2

[solution n°3 p. 31]

On se donne le script suivant :

```
1 let fruit = 'abricot'
2 let codeFruit = 104910
3 fruit = codeFruit
```

Modifier le code pour savoir si `fruit` et `codeFruit` référencent la même valeur.

Question 3

[solution n°4 p. 31]

On se donne le code JavaScript suivant :

```
1 const codeAbricot = 104910
2 const panierLundi = [codeAbricot]
3 const panierMardi = [codeAbricot]
```

Modifier le code pour savoir si `panierLundi` et `panierMardi` référencent la même valeur.

Question 4

[solution n°5 p. 32]

On se donne le code JavaScript suivant :

```
1 const codeAbricot = 104910
2 const panierLundi = [codeAbricot]
3 const panierMardi = [codeAbricot]
4 const premierFruitPanierLundi = panierLundi[0]
5 const premierFruitPanierMardi = panierMardi[0]
```

Modifier le code pour savoir si `premierFruitPanierLundi` et `premierFruitPanierMardi` référencent la même valeur.

V Clonage de variables de type composé

Objectifs

- Savoir copier des valeurs en JavaScript et en python ;
- Connaître la différence entre copie et copie récursive.

Mise en situation

Nous avons appris à copier une variable, et cela n'a rien de très compliqué. Cependant cela est valable pour les variables simples, et c'est une autre affaire quand il s'agit de variables composées, comme des enregistrements. Par exemple si l'on copie un tableau d'enregistrements, qui contiennent eux même des enregistrements, est-ce que les enregistrements imbriqués sont dupliqués ou est-ce que la copie du tableau ne fera que de simples références ? On sent bien que ici la copie des variables est un peu plus compliquée à appréhender.

Une erreur commune

Exemple

Parfois on souhaite copier des valeurs entre différentes variables. Pour cela on utilise généralement l'affectation. Par exemple en Python :

```python
tokens = [1, 2, 3, 4, 5]
numbers = tokens
print(numbers)
```

```
[1, 2, 3, 4, 5]
```

Néanmoins, si on modifie le « premier tableau », le « second » est aussi modifié :

```python
tokens[2] = 98
print(numbers)
```

```
[1, 2, 98, 4, 5]
```

On obtient le même comportement en JavaScript :

```javascript
let tokens = [1, 2, 3, 4, 5]
let numbers = tokens
tokens[2] = 98
console.log(numbers)
```

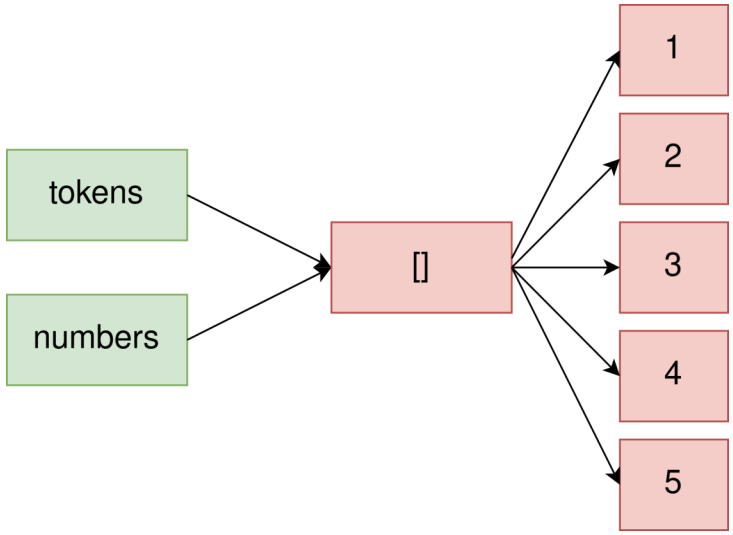
```
[1, 2, 98, 4, 5]
```

Pourquoi est-ce que cela se produit ? Comment y remédier ?

Explication : référencement de valeur de type composé

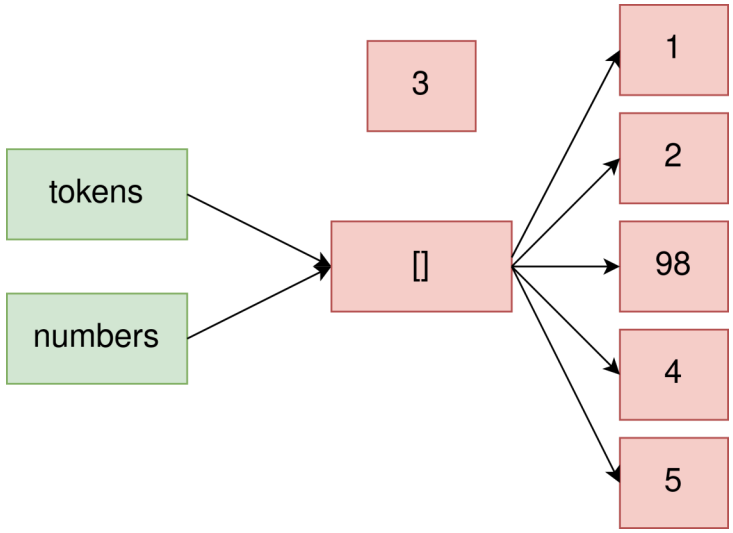
Fondamental

Lors de l'affectation d'une variable par une autre, on recopie dans le cas du type composé la référence vers la valeur.



Deux variables référençant le même tableau après affectation

Ainsi, si on change la variable à l'aide d'une des deux variables, l'autre variable renverra la valeur mise à jour et non l'ancienne.



Modification d'un élément d'un tableau référencé par deux variables

La valeur 3 n'est plus référencée : une nouvelle valeur 98 est créée, et référencée dans le tableau.

Explication de l'exemple précédent

Exemple

On peut voir que les deux variables en Python référencent la même valeur :

```
1 print(id(tokens) == id(numbers))
```

```
1 True
```

On peut voir cela aussi cela en JavaScript avec le test d'égalité avec l'opérateur ===.

```
1 print(tokens === numbers)
```

```
1 true
```

Réaliser une copie

Fondamental

Pour réaliser une copie d'une valeur d'un type composé il faut utiliser des fonctions dédiées qui s'occupent de réaliser la copie des variables de types primitifs.

Réaliser une copie en Python

Exemple

En Python on utilise la fonction `copy` du module `copy`.

```
1 import copy
2 tokens = [1, 2, 3, 4, 5]
3 numbers = copy.copy(tokens)
```

`numbers` et `tokens` référencent deux valeurs différentes :

```
1 print(id(tokens) == id(numbers))
```

```
1 False
```

Ainsi, si on modifie la valeur d'un variable l'autre n'est pas impactée :

```
1 tokens[2] = 98
2 print(tokens, numbers)
```

```
1 [1, 2, 98, 4, 5], [1, 2, 3, 4, 5]
```

Réaliser une copie en JavaScript

Exemple

En JavaScript, on utilise la méthode `Object.assign()` pour réaliser une copie.

```
1 let tokens = [1, 2, 3, 4, 5]
2 let numbers = Object.assign([], tokens)
```

Le comportement est identique à celui rencontré avec Python:

```
1 tokens[2] = 98
2 console.log(tokens, numbers)
```

```
1 [1, 2, 98, 4, 5], [1, 2, 3, 4, 5]
```

Copie récursive de valeurs

Complément

Lors d'une copie, seules les premières valeurs superficielles sont copiées, ainsi dans le cas de structures imbriquées (comme un tableau de tableau), toutes les valeurs ne sont pas copiées.

Pour recopier entièrement toutes les valeurs présentes, il faut utiliser une **copie récursive**.

Copie récursive de valeurs en Python

Exemple

En Python, on peut utiliser la fonction `copy.deepcopy` qui réalise une copie récursive de valeurs.

```
1 import copy
2
3 cours = {
4   'id_cours': 1337,
5   'nom_cours': 'IngDoc',
```

```
  'theme': 'Ingénierie Documentaire',
  'etudiants': [
    {
      'nom': 'Norris',
      'prenom': 'Chuck',
      'age': 73,
      'pays': 'USA'
    },
    {
      'nom': 'Doe',
      'prenom': 'Jane',
      'age': 45,
      'pays': 'Angleterre'
    }
  ]
}

copie_cours = copy.deepcopy(cours)
```

Copie récursive de valeurs en JavaScript

Remarque

En JavaScript (ES6), il n'est pas nativement possible de réaliser de copie récursive : il faut définir une fonction dédiée ou utiliser une bibliothèque spécialisée.

Copie récursive en JavaScript avec une fonction dédiée

Exemple

On définit la fonction suivante permettant de réaliser une copie récursive :

```
const deepCopyFunction = (inObject) => {
  let outObject, value, key

  if (typeof inObject !== "object" || inObject === null) {
   // Retourne la valeur de inObject si ce n'est pas un objet
   return inObject
  }

  // Crée un tableau ou un objet pour contenir les valeurs
  outObject = Array.isArray(inObject) ? [] : {}

  for (key in inObject) {
    value = inObject[key]

    // Copie récursivement les objets imbriquées, donc les tableaux
    outObject[key] = deepCopyFunction(value)
  }

  return outObject
}
```

```
const cours = {
  'id_cours': 1337,
  'nom_cours': 'IngDoc',
  'theme': 'Ingénierie Documentaire',
  'etudiants': [
    {
      'nom': 'Norris',
      'prenom': 'Chuck',
      'age': 73,
```

```
      'pays': 'USA'
    },
    {
      'nom': 'Doe',
      'prenom': 'Jane',
      'age': 45,
      'pays': 'Angleterre'
    }
  ]
}

const copie_cours = deepCopyFunction(cours)
cours.etudiants[0].id_cours = 1987
console.log(cours, copie_cours)
```

Ici, seul l'identifiant (`id`) du premier cours est modifié.

```
{
  id_cours: 1987,
  nom_cours: 'IngDoc',
  theme: 'Ingénierie Documentaire',
  etudiants: [
    { nom: 'Norris', prenom: 'Chuck', age: 73, pays: 'USA' },
    { nom: 'Doe', prenom: 'Jane', age: 45, pays: 'Angleterre' }
  ]
} {
  id_cours: 1337,
  nom_cours: 'IngDoc',
  theme: 'Ingénierie Documentaire',
  etudiants: [
    { nom: 'Norris', prenom: 'Chuck', age: 73, pays: 'USA' },
    { nom: 'Doe', prenom: 'Jane', age: 45, pays: 'Angleterre' }
  ]
}
```

Éléments de documentation de copie ⊕ Complément

Pour Python : https://docs.python.org/3/library/copy.html

Pour JavaScript : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign

Code source original de la fonction de copie récursive : https://gist.github.com/djD-REK/e8b1497e7fbf0374e4eada669e5609cf#file-custom-deep-copy-function-using-recursion-js

À retenir

Pour copier la valeur d'une variable dans une autre, il faut utiliser les fonctions de copie : `copy.copy` en python et `Object.assign` en JavaScript.

Pour réaliser une copie intégrale d'une valeur, il faut utiliser les fonctions ou syntaxes de copie récursive : `copy.deepcopy` en Python et mettre au point une fonction dédiée en JavaScript.

VI Exercice : Appliquer la notion

On se donne le script JavaScript suivant qui réalise la copie de données de panier sur un site en ligne :

```
const basket = {
  'Le Tour du monde en quatre-vingts jours': 53.90,
  'Les Misérables': 12.90,
  'À la recherche du temps perdu': 34.90
}

const secondBasket = basket
const thirdBasket = Object.assign({}, basket)
secondBasket['Le Tour du monde en quatre-vingts jours'] = 12.00
thirdBasket['À la recherche du temps perdu'] = 31.12
const fourthBasket = secondBasket
fourthBasket['Les Misérables'] = 7.90
```

Question

[solution n°6 p. 32]

Modifierz ce code pour afficher la valeur finale de `thirdBasket`.

Quelle est-elle ? Pourquoi ne subit-elle pas les modifications apportées à `secondBasket` et `fourthBasket` ?

VII Passage par valeur, passage par variable

Objectifs

- Connaître la notion de passage par valeur et de passage par référence ;
- Connaître le comportement en JavaScript et en Python.

Mise en situation

Selon comment un paramètre est passé à une fonction, le potentiel traitement de la fonction pourra avoir une incidence ou non sur la variable d'origine. En effet, si la variable passée en paramètre est copiée, toute modification effectuée par la fonction ne sera pas répercutée sur la variable d'origine. Et inversement si il s'agit d'une référence.

Il est important de bien maîtriser ces aspects lorsque l'on utilise des fonctions, car les deux comportements sont susceptibles de mener à des résultats très différents dans l'exécution du programme.

Passage par valeur

Az Définition

Le passage d'un paramètre par valeur **copie la valeur** du paramètre dans une variable **locale** à la fonction.

Ainsi, si l'argument est modifié, le paramètre passé à la fonction ne l'est pas.

Fondamental

Le passage des **types primitifs** en Python et en JavaScript s'apparente à un passage par valeur.

Passage par valeur en Python

Exemple

Voici un exemple de passage par valeur en Python.

```python
def doubleBankAccount(amount):
  amount = amount * 2

bankAccountAmount = 777
doubleBankAccount(bankAccountAmount)
print(bankAccountAmount)
```

```
777
```

Ici les valeurs ayant un type primitif ne sont pas modifiées suite à l'appel de la fonction.

Passage par valeur en JavaScript

Exemple

On obtient le même comportement en JavaScript.

```
function doubleBankAccount (amount) {
  amount = amount * 2
}
let bankAccountAmount = 777
doubleBankAccount(bankAccountAmount)
console.log(bankAccountAmount)

```

```
777
```

Passage par référence

Définition

Le passage d'un paramètre par référence **copie la référence** du paramètre dans un argument qui une variable locale à l'étendue de la fonction.

Ainsi, si l'argument est modifié, le paramètre passé à la fonction l'est aussi.

Fondamental

Le passage des **types composés** en Python et en JavaScript s'apparente à un passage par référence.

Passage par référence en JavaScript

Exemple

Voici un exemple de passage par référence en JavaScript.

```
function addSpecialNumber(numbers) {
  numbers.push(23)
}
let lottoNumbers = [32, 41, 130]
addSpecialNumber(lottoNumbers)
console.log(lottoNumbers)
```

```
[32, 41, 130, 23]
```

Ici `lottoNumber`, étant associé à une valeur d'un type composé est modifiée suite à l'appel de la fonction.

Passage par référence en Python

Exemple

On obtient le même comportement en Python.

```
def addSpecialNumber(numbers):
  numbers.append(23)

lottoNumber = [32, 41, 130]
addSpecialNumber(lottoNumber)
print(lottoNumber)
```

```
1 [32, 41, 130, 23]
```

Passage par assignation

Complément

En Python, le passage des arguments se fait plus exactement par affectation : l'affectation ne fait que créer des références aux objets et il n'y a pas d'alias entre le nom d'un argument dans l'étendue appelante et l'étendue appelée. Ainsi il n'y a donc pas d'appel par référence en soi.

Voir l'élément de documentation ici : https://docs.python.org/3/faq/programming.html#how-do-i-write-a-function-with-output-parameters-call-by-reference

À retenir

Un passage par valeur copie la valeur dans une variable de l'étendue d'une fonction. Un passage par référence copie la référence dans une variable de l'étendu d'une fonction.

En JavaScript et en Python, le passage par valeur s'applique sur les types primitifs, le passage par référence s'appliquer sur les types composés.

VIII Exercice : Appliquer la notion

On se donne le script JavaScript suivant :

```
1  function changeIdentity (age, names) {
2    age = age + 12
3    // Suppression des valeurs existantes
4    names.splice(0, names.length)
5    // Ajout de nouvelles valeurs
6    names.push('John')
7    names.push('Albert')
8    names.push('Smith')
9  }
10
11 let age = 23
12 let names = ['Luke', 'George', 'Simpsons']
13
14 changeIdentity(age, names)
```

Question 1

[solution n°7 p. 32]

Modifier le script pour obtenir les valeurs de `age` et de `names` à la fin de l'exécution du script.

Quelles sont-elles ?

Question 2

[solution n°8 p. 33]

Modifier le script pour obtenir les valeurs de `age` et de `names` à la fin de l'exécution de `changeIdentity`.

Quelles sont-elles ?

Question 3

[solution n°9 p. 33]

Comment expliquer les résultats précédents ?

IX Quiz

Exercice 1 : Quiz - Culture

[solution n°10 p. 33]

Exercice

Parmi les propositions suivantes, lesquelles sont associées à des types primitifs ?

- **A** Entier
- **B** Flottant
- **C** Chaîne de caractères
- **D** Tableau
- **E** Tableau associatif (dictionnaire en Python, objet en JavaScript)
- **F** Booléen

Exercice

Parmi les propositions suivantes, lesquelles sont associées à des types composés ?

- **A** Entier
- **B** Flottant
- **C** Chaîne de caractères
- **D** Tableau
- **E** Tableau associatif (dictionnaire en Python, objet en JavaScript)
- **F** Booléen

Exercice

Avec le typage dynamique, l'information du type est portée par :

- **A** la variable
- **B** la valeur

C la référence

Exercice 5 : Quiz - Méthode

[solution n°11 p. 34]

Exercice

Dans un langage au typage dynamique comme Python ou JavaScript, une variable change de référence lorsque l'on utilise une affectation, par exemple :

```
let first = 'Luke'
first = 'Campion'
```

A Vrai

B Faux

Exercice

Une affectation d'une variable a à une variable b... :

A donne à la variable b la référence contenue dans la variable a.

B déférence via b une nouvelle valeur dont le contenu est identique au contenu de la valeur de a.

Exercice

Dans le cas d'un type composé, une **copie** d'une variable a en une variable b, par exemple avec `Object.assign` en JavaScript ou `copy` en Python... :

A copie toujours l'entièreté des valeurs.

B ne copie parfois qu'une partie des valeurs.

Exercice 9 : Quiz - Code

[solution n°12 p. 35]

Exercice

Déterminer le résultat de l'exécution de ce programme.

```
animals = ['Porc', 'Vache', 'Poulet']
beings = animals
beings.pop()
console.log(animals)
```

A `['Porc']`

B `['Porc', 'Vache', 'Poulet']`

C `['Porc', 'Vache']`

Exercice

Qu'affiche le script suivant ?

```
let somebody = 'Pierre'
let somebodyelse = somebody
somebody = somebody + 'Paul'
somebodyelse = somebodyelse + 'Jacques'
console.log(somebody, somebodyelse)
```

Exercice

Qu'affiche le script suivant ?

```
function extendName(name, addon) {
  name = name + addon
  return name
}
let somebody = 'Pierre'
let somebodyelse = somebody
somebody = extendName (somebody, 'Paul')
somebodyelse = extendName (somebodyelse, 'Jacques')
console.log(somebody, somebodyelse)
```

Exercice

Qu'affiche le script suivant ?

```
function extendName(name, addon) {
  name.push(addon)
  return name
}
let somebody = ['Pierre']
let somebodyelse = somebody
somebody = extendName (somebody, 'Paul')
somebodyelse = extendName (somebodyelse, 'Jacques')
console.log(somebody.length, somebodyelse.length)
```

X Exercice : Défi

Émilie possède une collection de livres qu'elle souhaite vendre à un libraire en janvier. Elle dispose du prix d'achat de chacun des livres ainsi que d'une estimation du prix de vente sous la forme d'un tableau :

Titre livre	Prix d'achat (en €)	Prix de vente estimé (en €)
Les Fleurs du Mal	7.00	7.00
La Guerre des Intelligences	20.90	1.23
Inferno	9.99	5.30
...	...	...

Question 1

[solution n°13 p. 36]

Elle a réalisé un programme en JavaScript dont la logique amène à l'état suivant :

```
const bookListJanuaryEstimations =
[
  ['Les Fleurs du Mal', 7.00, 7.00],
  ['La Guerre des Intelligences', 20.90, 1.23],
  ['Inferno', 9.99, 5.30]
]

let totalCash = 0

for (const book of bookListJanuaryEstimations) {
  totalCash = totalCash + book[2]
}

console.log('Estimation de la vente en Janvier:', totalCash.toFixed(2), '€')
```

Combien d'argent peut-elle espérer gagner en Janvier avec la présente liste de livres ?

Question 2

[solution n°14 p. 37]

Afin de rendre la logique de calcul plus générale, Émilie décide de modifier le programme pour réaliser l'estimation de ses gains dans une fonction dédiée :

```
const bookListJanuaryEstimations =
[
  ['Les Fleurs du Mal', 7.00, 7.00],
  ['La Guerre des Intelligences', 20.90, 1.23],
  ['Inferno', 9.99, 5.30]
]

function estimateCashOnSell(totalCash, bookList) {
  for (const book of bookList) {
    totalCash = totalCash + book[2]
  }
}

let totalCash = 0

```

```
estimateCashOnSell(totalCash, bookListJanuaryEstimations)

console.log('Estimation de la vente en Janvier:', totalCash.toFixed(2), '€')
```

Néanmoins elle pense avoir réalisé une erreur. Quel est le résultat retourné par le programme ?

Pourquoi obtient-on ce résultat ?

Proposer une correction à apporter au programme pour corriger ce problème et vérifier qu'on obtient le même résultat que précédemment.

Question 3

[solution n°15 p. 37]

Émilie dispose maintenant d'estimation de la vente de ces livres pour le mois de février.

Titre livre	Prix d'achat (en €)	Prix de vente estimé (en €)
Les Fleurs du Mal	7.00	8.23
La Guerre des Intelligences	20.90	0.52
Inferno	9.99	5.30
…	…	…

Puisque peu de données changent, elle modifie son programme à la marge pour calculer l'estimation de ces gains sur ce nouveau mois ainsi :

```
const bookListJanuaryEstimations =
[
  ['Les Fleurs du Mal', 7.00, 7.00],
  ['La Guerre des Intelligences', 20.90, 1.23],
  ['Inferno', 9.99, 5.30]
]

bookListFebruaryEstimations = bookListJanuaryEstimations
bookListFebruaryEstimations[0][2] = 8.23
bookListFebruaryEstimations[1][2] = 0.52

function estimateCashOnSell(bookList) {
  let amountEstimate = 0
  for (const book of bookList) {
    amountEstimate = amountEstimate + book[2]
  }
  return amountEstimate
}


const totalCashJanuary = estimateCashOnSell(bookListJanuaryEstimations)
console.log('Estimation de la vente en Janvier:', totalCashJanuary.toFixed(2), '€')

const totalCashFebruary = estimateCashOnSell(bookListFebruaryEstimations)
console.log('Estimation de la vente en Février:', totalCashFebruary.toFixed(2), '€')
```

Néanmoins, elle trouve des résultats différents pour le mois de Janvier.

Quel est le résultat de l'exécution de ce programme ?

Pourquoi obtient-on un résultat différent sur le mois de Janvier comparé à précédemment ?

Question 4

[solution n°16 p. 38]

En relisant son code, elle s'est rendu compte qu'il s'agissait d'un problème de copie de tableaux. Elle modifie donc son code ainsi pour réaliser une copie avec la méthode `Object.assign`.

```
const bookListJanuaryEstimations =
[
  ['Les Fleurs du Mal', 7.00, 7.00],
  ['La Guerre des Intelligences', 20.90, 1.23],
  ['Inferno', 9.99, 5.30]
]

bookListFebruaryEstimations = Object.assign([], bookListJanuaryEstimations)
bookListFebruaryEstimations[0][2] = 8.23
bookListFebruaryEstimations[1][2] = 0.52

function estimateCashOnSell(bookList) {
  let amountEstimate = 0
  for (const book of bookList) {
    amountEstimate = amountEstimate + book[2]
  }
  return amountEstimate
}


const totalCashJanuary = estimateCashOnSell(bookListJanuaryEstimations)
console.log('Estimation de la vente en Janvier:', totalCashJanuary.toFixed(2), '€')

const totalCashFebruary = estimateCashOnSell(bookListFebruaryEstimations)
console.log('Estimation de la vente en Février:', totalCashFebruary.toFixed(2), '€')
```

Le résultat persiste-t-il ?

Pourquoi ? S'il persiste, comment peut-on résoudre ce problème ? Modifier le code en conséquence.

Indice :

On pourra utiliser la fonction de copie récursive suivante :

```
const deepCopy = (items) => items.map(item => Array.isArray(item) ? deepCopy(item) :
item)
```

Conclusion

Ce module nous a permis de voir différents concepts avancés sur l'utilisation des variables. Ce sont des concepts importants car ils touchent directement à la manière dont les variables sont manipulées en mémoire, et utilisées par des fonctions. Faire une copie ou une référence à une variable n'aura pas le même effet dans le programme, et une erreur à ce niveau là peut introduire des bugs importants.

Solutions des exercices

Solution n°1

[exercice p. 6]

On se donne le morceau de code suivant :

```
1 carNumbers = 1254358
2 moto = 'We do what we can because we must'
3 quote = moto
4 console.log('18126798')
5 console.log(24)
```

Associer les mots aux différentes cibles.

Variable	Valeur (entier)	Valeur (chaîne de caractères)
`moto`	`1254358`	`'We do what we can because we must'`
`quote`	`24`	`'18126798'`
`carNumbers`		

Solution n°2

[exercice p. 13]

```
1 prixAbricot = 4
2 prixPeche = 4
3 console.log(prixAbricot === prixPeche)
```

On obtient `true`, ce qui indique que les deux variables référencent la même valeur.

Solution n°3

[exercice p. 13]

```
1 let fruit = 'abricot'
2 let codeFruit = 104910
3 fruit = codeFruit
4 console.log(fruit === codeFruit)
```

On obtient `true`, ce qui indique que les deux variables référencent la même valeur, ici l'entier `104910`.

Solution n°4

[exercice p. 13]

```
1 const codeAbricot = 104910
2 const panierLundi = [codeAbricot]
3 const panierMardi = [codeAbricot]
4 console.log(panierLundi === panierMardi)
```

On obtient `false`, ce qui indique que les deux tableaux ne référencent pas la même valeur : il s'agit bien en effet de deux tableaux différents que l'on crée, même si chacun référence la même valeur 104910.

Solution n°5

[exercice p. 13]

```
const codeAbricot = 104910
const panierLundi = [codeAbricot]
const panierMardi = [codeAbricot]
const premierFruitPanierLundi = panierLundi[0]
const premierFruitPanierMardi = panierMardi[0]
console.log(premierFruitPanierLundi === premierFruitPanierMardi)
```

On obtient `true`, ce qui indique que les deux variables référencent la même valeur : il s'agit bien en effet de deux tableaux différents mais ils ont bien une même valeur en commun.

Solution n°6

[exercice p. 19]

Pour cela, on modifie le code pour ajouter :

```
console.log(thirdBasket)
```

On obtient :

```
let basket = {
  'Le Tour du monde en quatre-vingts jours': 53.90,
  'Les Misérables': 12.90,
  'À la recherche du temps perdu': 34.90
}

let secondBasket = basket
let thirdBasket = Object.assign({}, basket)
secondBasket['Le Tour du monde en quatre-vingts jours'] = 12.00
thirdBasket['À la recherche du temps perdu'] = 31.12
let fourthBasket = secondBasket
fourthBasket['Les Misérables'] = 7.90

console.log(thirdBasket)
```

`thirdBasket` a la valeur suivante :

```
{
 'Le Tour du monde en quatre-vingts jours': 53.9,
 'Les Misérables': 12.9,
 'À la recherche du temps perdu': 31.12
 }
```

Dans ce premier cas, `thirdBasket` référence une nouvelle structure via une copie de la valeur contenue dans `basket`.

Cette structure ne subit qu'un changement : celui via `thirdBasket` (modification du prix de `'À la recherche du temps perdu'`).

Elle ne subit pas de changements via les modifications réalisées sur `secondBasket` et sur `fourthBasket`.

Solution n°7

[exercice p. 23]

On ajoute l'instruction suivante à la fin du programme :

```
console.log(age, names)
```

On obtient les valeurs :

```
35, ['John', 'Albert', 'Smith']
```

Solution n°8

[exercice p. 23]

On ajoute l'instruction suivante à la fin de la **fonction** :

```
1 console.log(age, names)
```

On obtient les valeurs :

```
1 23, ['John', 'Albert', 'Smith']
```

Solution n°9

[exercice p. 23]

La variable `age` référence un type primitif, tandis que la variable `names` référence un type composé. Ainsi, dans et en dehors de la fonction :

- La variable `age` n'a **pas** la même référence ; ici la variable est **passée par valeur** puisque c'est un type primitif (un entier) : cela explique pourquoi sa valeur n'est pas affectée.
- La variable `names` a la même référence ; la variable est **passée par référence** puisque c'est un type composé (un tableau) : cela explique pourquoi sa valeur est changée.

Solution n°10

[exercice p. 24]

Exercice

Parmi les propositions suivantes, lesquelles sont associées à des types primitifs ?

- **A** Entier
- **B** Flottant
- **C** Chaîne de caractères
- **D** Tableau
- **E** Tableau associatif (dictionnaire en Python, objet en JavaScript)
- **F** Booléen

Exercice

Parmi les propositions suivantes, lesquelles sont associées à des types composés ?

- **A** Entier
- **B** Flottant
- **C** Chaîne de caractères
- **D** Tableau

E Tableau associatif (dictionnaire en Python, objet en JavaScript)

F Booléen

Exercice

Avec le typage dynamique, l'information du type est portée par :

A la variable

B la valeur

C la référence

Dans la mesure où une variable peut *« changer de type »*, l'information du type est portée par la **valeur** référencée par la variable.

Solution n°11

[exercice p. 25]

Exercice

Dans un langage au typage dynamique comme Python ou JavaScript, une variable change de référence lorsque l'on utilise une affectation, par exemple :

```
1 let first = 'Luke'
2 first = 'Campion'
```

A Vrai

B Faux

L'utilisation de l'opérateur d'affectation va systématiquement modifier la référence de la variable. Dans cet exemple, `first` référence `['Luke']`, puis référence `['Campion']`.

Exercice

Une affectation d'une variable a à une variable b... :

A donne à la variable b la référence contenue dans la variable a.

B déférence via b une nouvelle valeur dont le contenu est identique au contenu de la valeur de a.

L'affectation d'une variable à une autre va copier la référence de la première variable. Au moment de l'affectation, les deux variables référencent la même case mémoire, et non deux cases mémoires dont les valeurs seraient identiques.

Exercice

Dans le cas d'un type composé, une **copie** d'une variable a en une variable b, par exemple avec `Object.assign` en JavaScript ou `copy` en Python... :

A copie toujours l'entièreté des valeurs.

B ne copie parfois qu'une partie des valeurs.

Dans le cas des types composés seule une copie superficielle est appliquée. Il faut pour réaliser une copie entière des valeurs utiliser les outils de copie **récursive**, ou **profonde**.

Solution n°12

[exercice p. 25]

Exercice

Déterminer le résultat de l'exécution de ce programme.

```
1 animals = ['Porc', 'Vache', 'Poulet']
2 beings = animals
3 beings.pop()
4 console.log(animals)
```

A `['Porc']`

B `['Porc', 'Vache', 'Poulet']`

C `['Porc', 'Vache']`

Ici, `beings` et `animals` référencent le même tableau `['Porc', 'Vache', 'Poulet']`. À l'appel de la fonction `pop()`, ce tableau perd son dernier élément.

Exercice

Qu'affiche le script suivant ?

```
1 let somebody = 'Pierre'
2 let somebodyelse = somebody
3 somebody = somebody + 'Paul'
4 somebodyelse = somebodyelse + 'Jacques'
5 console.log(somebody, somebodyelse)
```

PierrePaul PierreJacques

somebody est une variable de valeur de type primitif :

- La modification de somebody ne modifie par somebodyelse.
- La modification de somebodyelse ne modifie par somebody.

Exercice

Qu'affiche le script suivant ?

```
function extendName(name, addon) {
  name = name + addon
  return name
}
let somebody = 'Pierre'
let somebodyelse = somebody
somebody = extendName (somebody, 'Paul')
somebodyelse = extendName (somebodyelse, 'Jacques')
console.log(somebody, somebodyelse)
```

PierrePaul PierreJacques

somebody est une variable de valeur de type primitif :

- La modification de somebody ne modifie par somebodyelse.
- La modification de somebodyelse ne modifie par somebody.

Exercice

Qu'affiche le script suivant ?

```
function extendName(name, addon) {
  name.push(addon)
  return name
}
let somebody = ['Pierre']
let somebodyelse = somebody
somebody = extendName (somebody, 'Paul')
somebodyelse = extendName (somebodyelse, 'Jacques')
console.log(somebody.length, somebodyelse.length)
```

3 3

somebody est une variable de valeur de type composé donc la modification de somebodyelse modifie somebody (elle référence le même tableau).

Solution n°13

[exercice p. 27]

On exécute le programme et on obtient :

```
Estimation de la vente en Janvier: 13.53 €
```

Solution n°14

[exercice p. 27]

On obtient :

```
Estimation de la vente en Janvier: 0.00 €
```

On obtient ce résultat car `totalCash` est passé par valeur : les ajouts qui sont réalisés n'influencent pas la valeur de la variable `totalCash` dans l'étendue globale.

Pour corriger ce problème, on peut corriger la fonction pour simplement retourner le montant total estimé de la vente et réaliser la somme avec `totalCash` ainsi :

```
const bookListJanuaryEstimations =
[
  ['Les Fleurs du Mal', 7.00, 7.00],
  ['La Guerre des Intelligences', 20.90, 1.23],
  ['Inferno', 9.99, 5.30]
]

function estimateCashOnSell(bookList) {
  let amountEstimate = 0
  for (const book of bookList) {
    amountEstimate = amountEstimate + book[2]
  }
  return amountEstimate
}

let totalCash = 0

totalCash = estimateCashOnSell(bookListJanuaryEstimations)

console.log('Estimation de la vente en Janvier:', totalCash.toFixed(2), '€')
```

On obtient alors le résultat attendu, à savoir :

```
Estimation de la vente en Janvier: 13.53 €
```

Solution n°15

[exercice p. 28]

Le résultat de l'exécution de ce programme est :

```
Estimation de la vente en Janvier: 14.05 €
Estimation de la vente en Février: 14.05 €
```

On obtient un résultat différent pour janvier car on a en réalité partagé une valeur de type composé (le tableau) entre les deux variables `bookListJanuaryEstimations` et `bookListFebruaryEstimations`. Ainsi les modifications faites via `bookListFebruaryEstimations` sont rendues visibles à partir de `bookListJanuaryEstimations`, d'où un résultat identique pour les deux mois et qui se base sur les prix estimés pour Février.

Solution n°16

[exercice p. 29]

Le résultat persiste, on obtient toujours :

```
Estimation de la vente en Janvier: 14.05 €
Estimation de la vente en Février: 14.05 €
```

La méthode `Object.assign` ne réalise qu'une copie superficielle du tableau : ainsi, les valeurs contenues dans les tableaux imbriqués ne sont pas recopiées et sont partagées entre les deux variables, et on rencontre le même problème que précédemment. Pour résoudre ce problème, c'est-à-dire pour recopier toutes les valeurs, il faut utiliser une copie récursive.

```
const deepCopy = (items) => items.map(item => Array.isArray(item) ? deepCopy(item) :
item)

const bookListJanuaryEstimations =
[
  ['Les Fleurs du Mal', 7.00, 7.00],
  ['La Guerre des Intelligences', 20.90, 1.23],
  ['Inferno', 9.99, 5.30]
]

bookListFebruaryEstimations = deepCopy(bookListJanuaryEstimations)
bookListFebruaryEstimations[0][2] = 8.23
bookListFebruaryEstimations[1][2] = 0.52


function estimateCashOnSell(bookList) {
  let amountEstimate = 0
  for (const book of bookList) {
    amountEstimate = amountEstimate + book[2]
  }
  return amountEstimate
}


const totalCashJanuary = estimateCashOnSell(bookListJanuaryEstimations)
console.log('Estimation de la vente en Janvier:', totalCashJanuary.toFixed(2), '€')

const totalCashFebruary = estimateCashOnSell(bookListFebruaryEstimations)
console.log('Estimation de la vente en Février:', totalCashFebruary.toFixed(2), '€')
```

On obtient bien le résultat souhaité :

```
Estimation de la vente en Janvier: 13.53 €
Estimation de la vente en Février: 14.05 €
```

Crédits des ressources

Référencement d'une valeur par une variable p. 5
Attribution - Partage dans les Mêmes Conditions - Quentin Duchemin

Référence d'une même valeur par deux variables p. 5
Attribution - Partage dans les Mêmes Conditions - Quentin Duchemin

Conservation de la référence lors de l'affectation d'une variable à une autre p. 9
Attribution - Partage dans les Mêmes Conditions - Quentin Duchemin

Copie de la valeur référencée lors de la modification d'un type primitif p. 11
Attribution - Partage dans les Mêmes Conditions - Quentin Duchemin

Conservation de la référence lors de la modification d'une valeur de type composé p. 12
Attribution - Partage dans les Mêmes Conditions - Quentin Duchemin

Deux variables référençant le même tableau après affectation p. 15
Attribution - Partage dans les Mêmes Conditions - Quentin Duchemin

Modification d'un élément d'un tableau référencé par deux variables p. 15
Attribution - Partage dans les Mêmes Conditions - Quentin Duchemin