Al Sweigart

Apprendre à coder des jeux vidéo en PYTHON

Dès 10 ans

no starch press

Éditions EYROLLES

Apprendre à coder des jeux vidéo en PYTHON

Ne vous contentez pas de jouer à des jeux, créez-les !

Cet ouvrage a pour ambition de vous initier au développement de jeux vidéo grâce au populaire langage Python, et ce, même si n'avez encore jamais programmé de votre vie !

Vous développerez d'abord des jeux classiques comme le Pendu, Devinez le nombre ou le Morpion, avant de vous attaquer à la conception de jeux plus avancés techniquement qui intègrent, entre autres, du texte, des animations graphiques et du son.

Par la même occasion, vous apprendrez les concepts de base de la programmation et des mathématiques pour amener vos compétences en codage de jeux vidéo à un autre niveau. Tous les projets de cet ouvrage sont basés sur la dernière version (3) de Python.

Au cours de votre lecture, vous allez acquérir des bases solides en matière de programmation Python. Quel nouveau jeu allez-vous ensuite pouvoir créer à l'aide de la puissance de Python ?

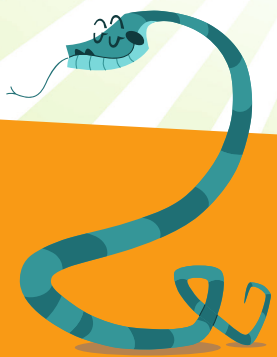
Tout au long de cette aventure en programmation, vous apprendrez à :

- choisir le bon type de structure de données pour faire le travail, comme des listes, dictionnaires ou tuples ;
- ajouter des illustrations et des animations dans votre jeu à l'aide du module pygame ;
- interagir avec le clavier et la souris ;
- programmer une intelligence artificielle suffisamment simple pour jouer contre l'ordinateur ;
- utiliser la cryptographie pour convertir des messages texte en codes secrets ;
- déboguer vos programmes et identifier les erreurs les plus communes.

À propos de l'auteur

Al Sweigart est un développeur de logiciels qui enseigne la programmation à la fois aux adultes et aux enfants. Il anime le blog inventwithpython.com qui propose de nombreux tutoriels de programmation.

Dans la même collection



www.editions-eyrolles.com

no starch press

Apprendre à coder des jeux vidéo en PYTHON

Al Sweigart

Apprendre à coder des jeux vidéo en PYTHON

no starch press

Éditions EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Traduction autorisée de l'ouvrage en langue anglaise intitulé

Invent Your Own Computer Games with Python, 4th Edition

de Al Sweigart (ISBN : 978-1-59327-795-6),

publié par No Starch Press.

TABLES DES MATIÈRES

7
L'organigramme du jeu du pendu 93

8
Écrire le code du pendu 105

12
Le système de coordonnées cartésiennes 191

13
La chasse au trésor 199

19
Détection de collision 323

20
Utiliser des sons et des images 339

INTRODUCTION

Lorsque j'ai commencé à jouer aux jeux vidéo en tant qu'enfant, j'ai été « scotché ». Cependant, je ne voulais pas uniquement jouer ; je voulais aussi créer mes jeux. J'ai alors trouvé un livre comme celui-ci pour apprendre à réaliser mes premiers programmes. C'était simple et amusant. Mes premiers jeux étaient semblables à ceux de cet ouvrage. Ils n'étaient pas du niveau de ceux de Nintendo que mes parents m'achetaient, mais c'est moi qui les avais conçus.

Maintenant que je suis adulte, j'ai toujours plaisir à programmer et je suis payé pour cela. Même si vous ne souhaitez pas en faire votre métier, il est bon de savoir programmer. Cela entraîne le cerveau à penser de façon logique, à concevoir des plans et à reconsidérer ses idées face à des erreurs dans le code.

De nombreux livres de programmation pour débutants tombent dans deux catégories. Dans la première, on trouve des ouvrages qui présentent la programmation de façon tellement

simplifiée que ce n'en est plus. Dans l'autre catégorie, on trouve des livres qui enseignent la programmation comme s'il s'agissait de mathématiques, avec tous les principes et les concepts, mais sans lien avec le monde réel. Le livre que vous lisez tente une approche différente et vous indique comment programmer en réalisant des jeux vidéo ; je montre le code source des jeux puis j'explique les principes à partir d'exemples. C'est ainsi que j'ai étudié la programmation. Plus j'ai appris sur la manière de travailler des autres, plus j'ai eu d'idées pour mes propres codes.

Tout ce dont vous avez besoin, c'est d'un ordinateur, d'un logiciel gratuit appelé un interpréteur Python et de cet ouvrage. Une fois que vous aurez appris à créer des jeux à l'aide de ce livre, vous serez en mesure d'écrire vos propres programmes.

Les ordinateurs sont des machines incroyables et, contrairement à ce que pensent beaucoup de personnes, ce n'est pas très compliqué de les commander. Un programme informatique est un ensemble d'instructions que l'ordinateur comprend, de la même manière qu'un roman est un ensemble de mots que le lecteur comprend. Pour donner des instructions à un ordinateur, on utilise un langage de programmation. Dans cet ouvrage, nous utilisons Python. Il existe de nombreux autres langages, comme Java, JavaScript, PHP ou C++.

Quand j'étais enfant, j'ai appris le BASIC, mais de nouveaux langages comme Python sont plus faciles à comprendre. Python est utilisé par des professionnels pour leur travail, mais aussi pour le divertissement. Ce langage est totalement gratuit et il suffit d'une connexion à Internet pour le télécharger.

Comme les jeux vidéo ne sont rien d'autre que des programmes, ils sont également constitués d'instructions. Les jeux que vous allez créer à partir de cet ouvrage semblent simples comparés à ceux pour Xbox, Playstation ou Nintendo. Ils ne présentent pas de jolis graphiques parce que le but est de vous apprendre les bases de la programmation. Ils sont volontairement épurés pour que vous vous concentriez sur votre apprentissage. De plus, les jeux n'ont pas à être complexes pour être divertissants.

À qui s'adresse ce livre ?

La programmation n'est pas compliquée. En revanche, il est compliqué de trouver des sujets pour l'enseigner de façon intéressante. Les ouvrages traitent souvent de nombreux sujets inutiles aux débutants. Ce livre va vous apprendre des techniques utiles en réalisant des jeux amusants. Il s'adresse aux personnes suivantes :

- débutants qui veulent apprendre la programmation, même s'ils n'ont aucune expérience ;
- enfants et adolescents qui veulent apprendre la programmation en créant des jeux ;
- adultes et professeurs qui souhaitent enseigner la programmation ;
- toute personne, jeune ou moins jeune, qui souhaite approfondir sa connaissance de la programmation avec un langage professionnel.

À propos de cet ouvrage

Dans la plupart des chapitres de ce livre, un jeu simple est présenté et expliqué. Quelques chapitres traitent de sujet utiles et complémentaires (comment déboguer un programme, par exemple). Les nouveaux concepts de programmation sont expliqués quand les jeux y font appel et les chapitres sont conçus pour être lus dans l'ordre.

- **Chapitre 1 – le shell interactif**

  Servez-vous du shell interactif de Python pour tester du code ligne après ligne.

- **Chapitre 2 – écrire des programmes**

  Utilisez l'éditeur de code de Python pour écrire des programmes.

- **Chapitre 3 – devinez le nombre**

  Écrivez votre premier jeu, qui demande au joueur de deviner un nombre et lui indique si sa proposition est supérieure ou inférieure. Maniez les entrées-sorties et les comparaisons de base.

- **Chapitre 4 – un programme de blagues**

  Manipulez les chaînes de caractères dans un programme simple qui raconte des blagues à l'utilisateur.

- **Chapitre 5 – l'antre du dragon**

  Définissez des fonctions. Le joueur doit choisir entre deux grottes : l'une d'elles est occupée par un gentil dragon et l'autre par un dragon affamé.

- **Chapitre 6 – utiliser le débogueur**

  Découvrez un outil très utile pour résoudre des problèmes dans votre programme.

- **Chapitre 7 – l'organigramme du pendu**

  Prenez l'habitude d'écrire des organigrammes pour planifier les longs programmes, comme celui du jeu du pendu.

- **Chapitre 8 – écrire le code du pendu**

  Découvrez les méthodes et les listes pour deviner un mot en dessinant un élément du pendu à chaque mauvaise réponse.

- **Chapitre 9 – améliorer le jeu du pendu**

  Ajoutez de nouvelles fonctionnalités au jeu précédent à l'aide du type de données dictionnaire de Python.

- **Chapitre 10 – le jeu du morpion**

  Programmez une intelligence artificielle (IA) pour faire jouer l'humain contre l'ordinateur.

- **Chapitre 11 – le jeu pico, fermi, bagels**

  Mélangez, triez et complétez des listes pour un jeu de déduction dans lequel le joueur doit devenir un nombre, guidé par des indices.

- **Chapitre 12 – le système de coordonnées cartésiennes**

  (Re)découvrez le système de coordonnées qui vous sera utile dans les chapitres suivants.

- **Chapitre 13 – la chasse au trésor**

  Définissez des structures de données complexes pour représenter un système de coordonnées. Appliquez-les dans un jeu consistant à trouver des trésors enfouis dans l'océan.

- **Chapitre 14 – le chiffrement de César**

  Allez plus loin dans la manipulation de chaînes de caractères pour chiffrer et déchiffrer des messages.

- **Chapitre 15 – Le jeu d'Othello**

  Codez un jeu ordinateur-contre-humain avancé, pratiquement imbattable, capable de simuler un choix et d'en évaluer les conséquences.

- **Chapitre 16 – simulation avec l'IA**

  Testez différents algorithmes en transformant le jeu d'Othello pour que plusieurs IA s'affrontent.

- **Chapitre 17 – créer des graphiques**

  Découvrez le module `pygame` de Python et son utilisation pour dessiner des graphiques en 2D.

- **Chapitre 18 – animer des graphiques**

  Animez des formes géométriques simples avec `pygame`.

- **Chapitre 19 – détection de collision**

  Détectez et traitez la collision d'objets dans les graphiques 2D.

- **Chapitre 20 – utiliser les sons et les images**

  Améliorez vos jeux en y intégrant des sons et des images.

- **Chapitre 21 – un jeu d'esquive avec images et sons**

  Combinez les concepts des chapitres 17 à 20 pour concevoir un programme d'évitement : Dodger.

1

LE SHELL INTERACTIF

Avant de créer des jeux, vous devez connaître les bases de la programmation. Dans ce chapitre, vous allez apprendre à utiliser le shell interactif de Python et effectuer des calculs de base.

Sujets traités dans ce chapitre

- Opérateurs
- Chiffres entiers et décimaux
- Valeurs
- Expressions
- Erreurs de syntaxe
- Stocker des valeurs dans des variables

Calculs mathématiques simples

Lancez IDLE en suivant les étapes de « Démarrer IDLE », page 8. Tout d'abord, nous allons utiliser Python pour résoudre quelques calculs mathématiques simples. Le shell interactif peut servir de calculatrice. Saisissez `2+2` à l'invite `>>>` puis appuyez sur **Entrée** (ou **Retour** selon les ordinateurs). La figure 1-1 montre la réponse, qui s'affiche immédiatement.

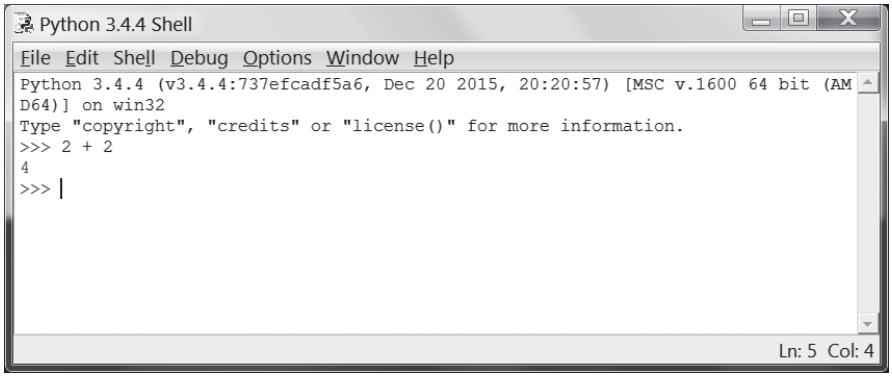


FIGURE 1-1 *Saisissez 2+2 dans le shell interactif.*

La résolution de ce problème mathématique se code à l'aide de quelques instructions de programmation. Le signe plus `+` indique à l'ordinateur d'ajouter les deux chiffres 2. L'ordinateur effectue l'opération et donne la réponse en affichant le chiffre 4 sur la ligne suivante. Le tableau 1-1 liste les autres opérateurs mathématiques disponibles dans Python.

TABLEAU 1-1 Opérateurs mathématiques

Opérateur	Opération
`+`	Addition
`-`	Soustraction
`*`	Multiplication
`/`	Division

Le signe `-` soustrait des nombres, l'astérisque `*` les multiplie et la barre de fraction `/` les divise. Lorsqu'ils sont utilisés pour ces fonctions, ces symboles sont appelés des opérateurs. Les opérateurs indiquent à Python ce qu'il faut faire avec les chiffres qui les entourent.

Entiers et décimaux

Les entiers sont des nombres sans décimales comme `4`, `99` ou `0`. Les décimaux sont des nombres avec au moins une décimale comme `3.5`, `42.1` ou `5.0`. Dans Python, `5` est un nombre entier mais `5.0` est un nombre à décimale. Notez qu'en programmation on utilise la notation anglo-saxonne : le séparateur décimal est un point et non pas une virgule.

Ces nombres sont appelés des valeurs (par la suite, vous découvrirez d'autres valeurs en plus des nombres). Dans le problème de mathématique que vous avez saisi dans le shell, `2` et `2` sont des valeurs entières.

Expressions

Le problème mathématique `2+2` est l'exemple d'une expression. Comme le montre la figure 1-2, les expressions sont composées de valeurs (les nombres) connectées par des opérateurs et produisent une nouvelle valeur (l'expression) que le code peut utiliser. Les ordinateurs savent résoudre des millions d'expressions par seconde.

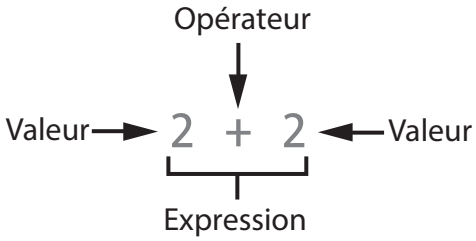


FIGURE 1-2 *Une expression est composée de valeurs et d'opérateurs.*

Essayez de saisir les expressions suivantes dans le shell interactif, en appuyant sur **Entrée** après chacune des expressions.

```
>>> 2+2+2+2+2
10
>>> 8*6
48
>>> 10-5+6
11
>>> 2 +          2
4
```

Notez les espaces dans le dernier exemple. Avec Python, vous pouvez insérer n'importe quel nombre d'espaces entre les valeurs et les opérateurs. Toutefois, vous devez toujours démarrer l'expression au début de la ligne (sans espace) lorsque vous utilisez le mode shell interactif.

Évaluer les expressions

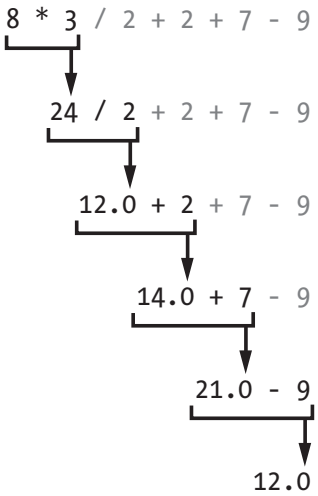
Lorsqu'un ordinateur résout `10+5` et renvoie la valeur `15`, on dit qu'il a évalué l'expression. Évaluer une expression revient à la réduire à une simple valeur, de la même manière que la résolution d'un problème de mathématique réduit ce dernier à une simple valeur : la réponse. Par exemple, les expressions `10+5` et `10+3+2` renvoient la valeur `15` lorsqu'elles sont évaluées.

Lorsque Python évalue une expression, il suit un ordre pour les opérations, de la même manière qu'en mathématiques. Voici quelques règles.

- Les éléments de l'expression se trouvant entre parenthèses sont évalués en premier.
- Les multiplications et les divisions sont réalisées avant les additions et les soustractions.
- L'évaluation s'effectue de gauche à droite.

L'évaluation de l'expression `1+2*3+4` donne `11` et non pas `13`, parce que l'opération `2*3` est évaluée en premier. Si l'expression avait été écrite `(1+2)*(3+4)`, le résultat aurait été `21`, parce que les opérations entre parenthèses `(1+2)` et `(3+4)` sont évaluées avant la multiplication.

Les expressions peuvent avoir n'importe quelle longueur mais elles ont toujours comme résultat une valeur unique. Même les valeurs uniques sont des expressions. Par exemple, l'évaluation de l'expression `15` donne la valeur `15`. L'expression `8*3/2+2+7-9` donne la valeur `12.0` à l'aide des étapes suivantes :



Même si l'ordinateur effectue les étapes précédentes, vous ne le voyez pas en mode shell interactif. Ce dernier vous renvoie uniquement le résultat.

```
>>> 8 * 3 / 2 + 2 + 7 - 9
12.0
```

Notez que les expressions contenant l'opérateur de division renvoient toujours une valeur décimale. Par exemple, `24/2` donne `12.0`. Les opérations mathématiques avec au moins une valeur décimale renvoient également une valeur décimale. Ainsi, `12.0+2` donne `14.0`.

Erreurs de syntaxe

Si vous saisissez `5+` dans le shell interactif, vous obtenez le message d'erreur suivant :

```
>>> 5 +
SyntaxError: invalid syntax
```

Cette erreur apparaît parce que `5+` n'est pas une expression. Les expressions sont composées de valeurs connectées par des opérateurs et l'opérateur `+` nécessite une valeur avant et une valeur après. Un message d'erreur apparaît lorsqu'une valeur attendue a été oubliée.

`SyntaxError` indique que Python ne comprend pas l'expression parce que vous ne l'avez pas saisie correctement. En programmation, il ne suffit pas de donner des instructions à l'ordinateur ; il est nécessaire de savoir comment les donner correctement.

Ne vous inquiétez pas de faire des erreurs. Celles-ci n'endommageront pas votre ordinateur. Il suffit de saisir à nouveau l'expression dans le mode shell interactif après l'invite.

Stocker les valeurs dans des variables

Lorsqu'une expression renvoie une valeur, vous pouvez utiliser cette dernière par la suite en la stockant dans une variable. Imaginez une variable comme une boîte dans laquelle vous placez la valeur.

Un opérateur d'affectation stocke une valeur dans une variable. Saisissez un nom pour la variable, suivi du signe égal (`=`), qui est l'opérateur d'affectation. La valeur est alors stockée dans la variable. Par exemple, saisissez ceci dans le shell interactif :

```
>>> spam = 15
>>>
```

La boîte `spam` contient maintenant la valeur `15` comme le montre la figure 1-3.

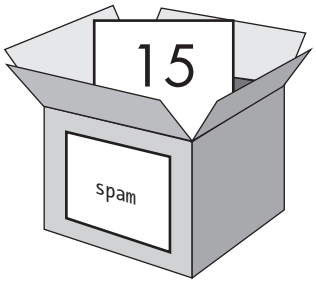


FIGURE 1-3 *Les variables sont comme des boîtes qui peuvent contenir des valeurs.*

Lorsque vous appuyez sur **Entrée**, vous n'obtenez pas de résultat. Avec Python, vous savez que l'instruction était correcte si aucun message d'erreur ne s'affiche. L'invite apparaît afin de vous indiquer que vous pouvez saisir l'instruction suivante.

Contrairement aux expressions, les affectations sont des instructions qui ne renvoient pas de valeur. C'est pour cela qu'aucune valeur n'apparaît dans le shell interactif lorsque vous saisissez `spam=15`. Si vous avez du mal à faire la différence entre les expressions et les affectations, gardez à l'esprit que les expressions renvoient une valeur. Tout autre type d'instruction est une affectation.

Les variables stockent des valeurs et non pas des expressions. Prenons, par exemple, les expressions `spam=10+5` et `spam=10+7-2`. L'éva-

luation donne le même résultat dans les deux cas et l'opérateur d'affectation stocke la valeur `15` dans la variable `spam`.

Un nom de variable est correct lorsqu'il décrit les données contenues. Imaginez que vous changez de maison et que vous marquez `Trucs` sur tous vos cartons de déménagement. Vous ne retrouverez jamais rien ! Les noms de variables, `spam`, `eggs` et `bacon` sont des noms utilisés pour les variables dans cet ouvrage.

La première fois qu'une variable est utilisée avec un opérateur d'affectation, Python la crée. Pour connaître sa valeur, saisissez son nom dans le shell interactif.

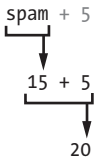
```
>>> spam = 15
>>> spam
15
```

L'expression `spam` est évaluée et on obtient son contenu : `15`.

Vous pouvez aussi utiliser des variables dans les expressions. Essayez de saisir ce qui suit dans le shell interactif :

```
>>> spam = 15
>>> spam + 5
20
```

Vous avez indiqué que la valeur de la variable `spam` est `15`. Donc, saisir `spam+5` revient à saisir `15+5`. Voici les étapes montrant comment l'expression `spam+5` est évaluée :



Vous ne pouvez pas utiliser une variable avant qu'un opérateur d'affectation ne la crée. Sinon, Python va renvoyer l'erreur `NameError` parce que la variable n'existe pas. Si vous n'orthographiez pas correctement le nom d'une variable, vous obtenez le même message d'erreur :

```
>>> spam = 15
>>> spma
Traceback (most recent call last):
File "<pyshell#8>", line 1, in <module> spma
NameError: name 'spma' is not defined
```

L'erreur s'affiche parce qu'il existe une variable `spam` mais pas de variable `spma`.

Vous modifiez la valeur stockée dans une variable en saisissant un autre opérateur d'affectation. Par exemple, saisissez ceci dans le shell interactif :

```
>>> spam = 15
>>> spam + 5
20
>>> spam = 3
>>> spam + 5
8
```

Lorsque vous saisissez une première fois `spam+5`, le résultat donne `20` parce que vous aviez stocké la valeur `15` dans `spam`. Toutefois, lorsque vous avez saisi `spam=3`, la valeur `15` a été remplacée ou écrasée avec la valeur `3`. Lorsque vous saisissez à nouveau `spam+5`, le résultat est `8`. Lorsque vous donnez une nouvelle valeur à une variable, c'est comme si vous retiriez l'ancienne de la boîte pour en placer une autre, comme le montre la figure 1-4.

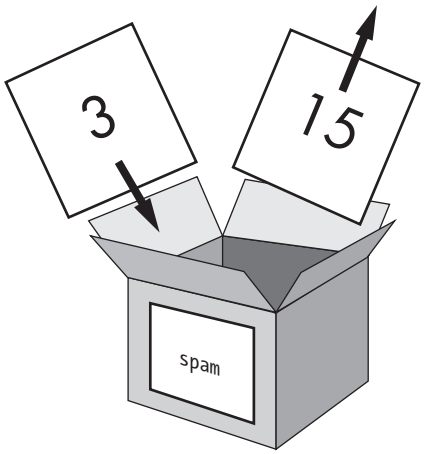


FIGURE 1-4 *La valeur 15 est remplacée par la valeur 3.*

Vous pouvez même utiliser la valeur contenue dans `spam` pour la modifier :

```
>>> spam = 15
>>> spam = spam + 5
20
```

L'opérateur d'affection `spam=spam+5` dit « la nouvelle valeur de `spam` sera la valeur actuelle plus cinq ». Pour ajouter plusieurs fois la valeur `5` à `spam`, saisissez ce qui suit dans le shell interactif :

```
>>> spam = 15
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam = spam + 5
>>> spam
30
```

Dans cet exemple, vous donnez initialement à `spam` la valeur `15`. À la ligne suivante, vous ajoutez `5` ; `spam` vaut donc maintenant `20`. En répétant cela trois fois, `spam` prend `30` comme valeur.

Jusqu'à présent, nous avons utilisé une seule variable, mais vous en créerez autant que nécessaire pour vos programmes. Par exemple, donnons des valeurs différentes à deux variables nommées `truc` et `bidule` :

```
>>> truc = 20
>>> bidule = 15
```

Maintenant, la variable `truc` vaut `20` et la variable `bidule` vaut `15`. Chacune des valeurs se trouve dans sa propre boîte, comme le montre la figure 1-5.

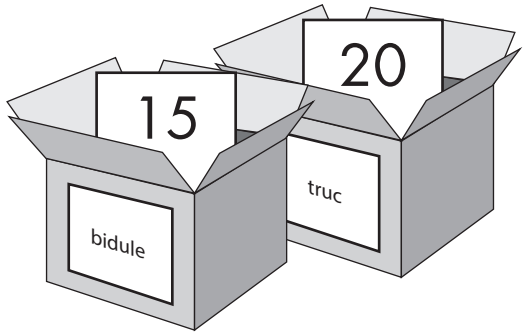


FIGURE 1-5 *Les variables truc et bidule stockent chacune une valeur.*

Saisissez `spam=truc+bidule` dans le shell interactif, puis affichez la nouvelle valeur de `spam` :

```
>>> spam = truc + bidule
>>> spam
35
```