La Pensée Python, Comment raisonner comme un scientiste de l'informatique

traduction du livre:

Think Python, How To Think Like A Computer Scientist

d'Allan Downey

- par Abdur-Rahmaan Janhangeer de l'Ile Maurice

Indexe

Chapitre 1

Le chemin de la programmation

Le but de ce livre est de vous apprendre à penser comme un informaticien. Cette façon de penser combine certaines des meilleures caractéristiques des maths, de l'ingénierie et des sciences naturelles. Comme les mathématiciens, les informaticiens utilisent des langages formels pour désigner des idées (en particulier les calculs). Comme les ingénieurs, ils conçoivent des choses, assemblent des composants dans des systèmes et évaluent les compromis entre les alternatives. Comme les scientifiques, ils observent le comportement des systèmes complexes, forment des hypothèses et testent des prédictions.

La compétence la plus importante pour un informaticien est la résolution de problèmes. La résolution des problèmes signifie la capacité de formuler des problèmes, réfléchir de manière créative aux solutions et exprimer une solution claire et précise. Dans l'affirmative, le processus d'apprentissage au programmation est une excellente occasion d'affiner ses compétences en matière de résolution de problèmes. C'est pourquoi ce chapitre s'appelle, "Le chemin de la programmation".

À un niveau, vous apprendrez à programmer, une compétence utile en soi. Sur un autre niveau, vous utiliserez la programmation comme un moyen d'atteindre une fin. À mesure que nous progressons, cette fin deviendra plus claire.

1.1 Qu'est-ce qu'un programme?

Un programme est une séquence d'instructions qui spécifie comment effectuer un calcul. Le calcul peut être quelque chose de mathématique, comme la résolution d'un système d'équations ou trouver les racines d'un polynôme, mais il peut également s'agir d'un calcul symbolique, comme la recherche, remplacer du texte dans un document ou quelque chose de graphique, comme le traitement d'une image ou jouer une vidéo.

Les détails sont différents dépendant des langues, mais quelques instructions de base apparaissent dans la plupart des langues:

- Entrée: Obtenez des données du clavier, d'un fichier, du réseau ou d'un autre appareil.
- Sortie: afficher les données sur l'écran, l'enregistrer dans un fichier, envoie sur réseau, etc.
- Maths: effectuez des opérations mathématiques de base comme addition et multiplication.
- Exécution conditionnelle: vérifier certaines conditions et exécuter le code approprié.
- Répétition: Effectuez une action répétée, habituellement avec une certaine variation.

Croyez-le ou non, c'est à peu près tout ce qu'il y a à faire. Chaque programme que vous avez déjà utilisé, quelle que soit la complexité, se compose d'instructions qui ressemblent à peu près à celles-ci. Ainsi, vous pouvez penser à la programmation en tant que processus de rupture d'une tâche complexe en de sous-tâches plus petites et plus petites jusqu'à ce que les sous-tâches soient assez simples pour être exécutées avec une de ces instructions de base.

1.2 Exécuter Python

L'un des défis de commencer avec Python est que vous devrez peut-être installer Python et logiciels relatifs sur votre ordinateur. Si vous connaissez votre système d'exploitation, et surtout si vous êtes à l'aise avec l'interface de ligne de commande, vous n'aurez aucun problème pour installer Python. Mais pour les débutants, il peut être douloureux d'apprendre l'administration du système et la programmation en même temps.

Pour éviter ce problème, je vous recommande de commencer à exécuter Python dans un navigateur. Plus tard, lorsque vous êtes à l'aise avec Python, je ferai des suggestions pour installer Python sur votre ordinateur.

Il existe un certain nombre de pages Web que vous pouvez utiliser pour exécuter Python. Si vous avez déjà un favori, allez-y et utilisez-le. Sinon, je recommande PythonAnywhere. Voyez les instructions sur http://tinyurl.com/thinkpython2e

Il existe deux versions de Python, appelées Python 2 et Python 3. Elles sont très similaires, donc si vous en apprendrez un, il est facile de passer à l'autre. En fait, il n'y a que quelques différences que vous rencontrerez en tant que débutant. Ce livre est écrit pour Python 3, mais j'ai inclu des notes sur Python 2.

L'interpréteur Python est un programme qui lit et exécute le code Python. En fonction de votre environnement, vous pouvez lancer l'interprète en cliquant sur une icône ou en tapant python sur une ligne de commande. Quand cela commence, vous devriez voir:

```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Les trois premières lignes contiennent des informations sur l'interprète et le système d'exploitation actuel, donc il pourrait être différent pour vous. Mais vous devez vérifier que le numéro de version, qui est 3.4.0 dans cet exemple, commence par 3, ce qui indique que vous utilisez Python 3. Si cela commence avec 2, vous exécutez (vous l'avez deviné) Python 2.

La dernière ligne indique que l'interprète est prêt à recevoir le code. Si vous tapez une ligne de code et appuyez sur Entrée (sur le clavier), l'interprète affiche le résultat:

```
>>> 1 + 1
2
```

Maintenant, vous êtes prêt à commencer. À partir de là, je suppose que vous savez comment commencer l'interprète de Python et exécuter des lignes de code.

1.3 Le premier programme

Traditionnellement, le premier programme que vous écrivez dans une nouvelle langue s'appelle "Hello, World!" Parce que

tout ce qu'il fait, c'est d'afficher les mots "Hello, World!". Dans Python, cela ressemble à ceci:

```
>>> print('Hello, World!')
```

Il s'agit d'un exemple d'une déclaration d'impression, bien qu'il n'imprime en réalité rien sur papier. Il affiche un résultat sur l'écran. Dans ce cas, le résultat est les mots

```
Hello, World!
```

Les guillemets dans le programme marquent le début et la fin du texte à afficher ; ils n'apparaissent pas dans le résultat.

Les parenthèses indiquent que l'impression est une fonction. Nous aborderons les fonctions du chapitre 3.

Dans Python 2, l'instruction d'impression est légèrement différente; ce n'est pas une fonction, donc les parenthèses, ça ne sert à rien.

```
>>> print 'Hello, World!'
```

Cette distinction aura plus de sens bientôt, mais c'est assez pour commencer.

1.4 Opérateurs arithmétiques

Après `" Hello, World!"`, la prochaine étape est l'arithmétique. Python fournit aux opérateurs, qui sont des symboles spéciaux qui représentent des calculs comme addition et multiplication.

Les opérateurs +, -, et * effectuent l'addition, la soustraction et la multiplication, comme suit exemples:

```
>>> 40 + 2
42
>>> 43 - 1
42
>>> 6 * 7
42

#L'opérateur / exécute la division:
>>> 84/2
42,0
```

Vous pourriez vous demander pourquoi le résultat est 42.0 au lieu de 42. Je vais vous expliquer dans la prochaine section.

Enfin, l'opérateur ** effectue une exponentiation; c'est-à-dire qu'il soulève un nombre à une puissance:

```
>>> 6 ** 2 + 6
42
```

Dans certaines autres langues, ^ est utilisé pour l'exponentiation, mais en Python, c'est un opérateur bit appelé XOR. Si vous n'êtes pas familiarisé avec les opérateurs bit, le résultat vous surprendra:

```
>>> 6 ^ 2
4
```

Je ne couvrirai pas les opérateurs bit à bit dans ce livre, mais vous pouvez les lire sur http://wiki.python.org/moin/BitwiseOperators.

1.5 Valeurs et types

Une valeur est l'une des choses de base avec laquelle un programme fonctionne, comme une lettre ou un numéro. Certains des valeurs que nous avons vues jusqu'ici sont `2`, `42.0`, et `'Hello, World!'`.

Ces valeurs appartiennent à différents types: `2` est un nombre entier, `42.0` est un float alias nombre décimal et `'Hello, World!'` est une chaîne de charactères (string en anglais), soi-disant parce que les lettres qu'il contient sont enfilées ensemble.

Si vous ne savez pas de quel type est une valeur, l'interprète peut vous dire:

```
>>> type (2)
<classe 'int'>
>>> type (42.0)
<classe 'float'>
>>> type (' Hello, World!')
<classe 'str'>
```

Dans ces résultats, le mot `«class»` est utilisé au sens d'une catégorie; un type est une catégorie de valeurs.

Il n'est pas surprenant que les entiers appartiennent au type `int`, les chaînes appartiennent à `str` et les nombres à virgule flottante à `float`.

Qu'en est-il des valeurs comme `'2'` et `'42 .0 '`? Ils ressemblent à des chiffres, mais ils sont en citations comme les strings.

```
>>> type ('2')
<classe 'str'>
>>> type ('42 .0 ')
<classe 'str'>
Ils sont des strings.
```

Lorsque vous tapez un grand nombre entier, vous pourriez être tenté d'utiliser des virgules entre les groupes de chiffres, soit 1 000 000. Ce n'est pas un entier juridiquement parlant en Python, mais c'est légal :

```
>>> 1,000,000
(1, 0, 0)
```

Ce n'est pas ce à quoi nous nous attendions du tout! Python interprète 1,000,000 comme une séquence d'entiers séparé par des virgules. Nous en apprendrons d'avantage sur ce genre de séquence plus tard.

1.6 Langues formelles et naturelles

Les langues sont les langues que les gens parlent, comme l'anglais, l'espagnol et le français. Ils n'étaient pas conçus par les gens (bien que les gens essayent d'imposer un ordre sur eux); ils évoluent naturellement.

Les langues formelles sont des langages conçus par des personnes pour des applications spécifiques. Par exemple, la notation utilisée par les mathématiciens est un langage formel qui est en particulier bien apte à dénoter les relations entre les nombres et les symboles. Les chimistes utilisent une langue formelle pour représenter la structure moléculaire des atomes. Et, surtout:

Les langages de programmation sont des langages formels conçus pour exprimer les calculs.

Les langues formelles ont tendance à avoir des règles de syntaxe strictes qui régissent la structure des déclarations. Par exemple, en mathématiques, la déclaration `3 + 3 = 6` a une syntaxe correcte, mais `3+ = 3 $ 6` ne l'est pas. En chimie H_2O est une formule syntaxiquement correcte, mais $_2Zz$ ne l'est pas.

Les règles de syntaxe comportent deux saveurs, relatives aux tokens et à la structure. Les tokens sont les éléments basiques de la langue, tels que les mots, les chiffres et les éléments chimiques. Un des problèmes avec `3+ = 3 $ 6` est que `$` n'est pas un token juridique en mathématiques (du moins à ce que je sais). De même, $_2Zz$ n'est pas légal car il n'y a aucun élément avec l'abréviation Zz.

Le second type de règle de syntaxe concerne la façon dont les tokens sont combinés. L'équation `3 += 3 est illégal` car même si + et = sont des tokens légaux, vous ne pouvez pas avoir un après l'autre. De même, dans une formule chimique, l'indice vient après le nom de l'élément, pas avant.

Ceci une phr@se bien $tructurée avec t * kens invalide. Cette phrase valable tokens a, mais structure invalide avec.

Lorsque vous lisez une phrase ou une déclaration dans une langue officielle, vous devez identifier la structure (bien que dans une langue naturelle, vous faites cela de façon inconsciente). Ce processus s'appelle parsing (analyse).

Bien que les langages formels et naturels possèdent de nombreuses caractéristiques communs : les token, la structure, et la syntaxe - il y a des différences:

- **ambiguïté:** les langues naturelles sont pleines d'ambiguïté, auxquelles les gens s'occupent en utilisant des indices contextuels et d'autres informations. Les langues officielles sont conçues pour être presque ou totalement sans ambiguïté, ce qui signifie que toute déclaration a exactement un sens, quel que soit le contexte.

- **redondance:** afin de compenser l'ambiguïté et de réduire les malentendus, les langues naturelles utilisent beaucoup de redondance. En conséquence, ils sont souvent détaillés. Les langues formelles sont moins redondantes et plus concises.

- **littéralité:** les langues naturelles sont pleines d'idiome et de métaphore. Si je dis en anglais: "The penny dropped" (litt. le penny est tombé), il n'y a probablement pas de penny et rien n'est tombé (cette idiome signifie que quelqu'un a compris quelque chose après une période de confusion). Les langues formelles signifient exactement ce qu'ils disent. Parce que nous grandissons tous en parlant des langues naturelles, il est parfois difficile de s'adapter à la forme langues. La différence entre langage formel et naturel est comme la différence entre la poésie et la prose, mais plus encore:

- **Poésie:** les mots sont utilisés pour leurs sons aussi bien que pour leur signification, et le poème entier ensemble crée un effet ou une réponse émotionnelle. L'ambiguïté n'est pas seulement souvent mais aussi délibéré.

- **Prose:** le sens littéral des mots est plus important, et la structure contribue davantage au sens. La prose est plus susceptible d'analyse que le poésie, mais toujours ambiguë.

- **Programmes:** la signification d'un programme informatique est sans ambiguïté et littérale, et peut être entièrement compris par l'analyse des tokens et de la structure.

Les langues formelles sont plus denses que les langues naturelles, il faut donc plus de temps pour les lire. En outre, la structure est importante, donc il n'est pas toujours préférable de lire de haut en bas, de gauche à droite. Au lieu de cela, apprenez à analyser le programme dans votre tête, identifiant les tokens et l'interprétation de la structure. Enfin, les détails sont importants. De petites erreurs dans l'orthographe et la ponctuation, dont vous pouvez vous en sortir dans des langues naturelles, peut faire une grande différence dans une langue formelle.

1.7 Débogage

Les programmeurs font des erreurs. Pour des raisons capricieuses, les erreurs de programmation sont appelées bugs et le processus de suivi est appelé débogage. La programmation, et en particulier le débogage, soulève parfois de fortes émotions. Si vous êtes en face d'un bug difficile, vous pouvez vous sentir en colère, découragé ou embarrassé.

Il existe des preuves que les gens répondent naturellement aux ordinateurs comme s'ils étaient des gens. Quand ils fonctionnent bien, on les considère comme des coéquipiers, et quand ils sont obstinés ou grossiers, nous répondons à eux de la même manière que nous répondons à des personnes grossières et obstinées (Reeves and Nass, *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*) .

Se préparer à ces réactions pourrait vous aider à les traiter. Une approche consiste à penser à l'ordinateur en tant qu'employé avec certaines forces, comme la vitesse et la précision, et en particulier les faiblesses, comme le manque d'empathie et l'incapacité de saisir l'image en gros.

Votre travail consiste à être un bon gestionnaire: trouver des moyens de tirer parti des atouts et atténuer les faiblesses. Et trouver des façons d'utiliser vos émotions pour s'engager dans le problème, sans laisser vos réactions interférer avec votre capacité à travailler efficacement.

Apprendre à déboguer peut être frustrant, mais c'est une compétence précieuse qui est utile pour de nombreuses activités au-delà de la programmation. À la fin de chaque chapitre, il y a une section, comme celle-ci, avec mes suggestions de débogage. J'espère qu'ils vous seront utiles!

1.8 Glossaire

- **résolution de problèmes / problem-solving:** le processus de formulation d'un problème, la recherche d'une solution et l'exprimer.
- **langage haut niveau / high-level language:** un langage de programmation comme Python conçu pour être facile pour les humains à lire et à écrire.
- **langage de bas niveau / low-level language:** un langage de programmation conçu pour être facile pour un ordinateur de parcourir; également appelé "language machine" ou "langage assembly".
- **portabilité:** une propriété d'un programme qui peut fonctionner sur plus d'un type d'ordinateur.
- **interprète:** un programme qui lit un autre programme et l'exécute
- **prompt:** caractères affichés par l'interprète pour indiquer qu'il est prêt à prendre une entrée de l'utilisateur.
- **programme:** un ensemble d'instructions qui spécifient un calcul.
- **print statement:** une instruction qui amène l'interpréteur Python à afficher une valeur sur l'écran.
- **opérateur:** un symbole spécial qui représente un calcul simple comme : addition, multiplication ou la concaténation de chaîne de caractères .
- **valeur:** une des unités de base de données, comme un nombre ou une chaîne, qu'un programme manipule.
- **type:** une catégorie de valeurs. Les types que nous avons vus jusqu'ici sont des entiers (type int), les nombres flottant (type float) et les chaînes (type str).
- **int:** un type qui représente des nombres entiers.
- **float:** un type qui représente des nombres décimaux.
- **string:** un type qui représente des séquences de caractères. Langue naturelle: l'une des langues que les gens parlent et qui évolue naturellement.
- **langage formel:** l'une des langues que les gens ont conçues à des fins spécifiques, tels que la représentation d'idées mathématiques ou de programmes informatiques; toute les langues de programmation sont des langues officielles.
- **token:** l'un des éléments de base de la structure syntaxique d'un programme, analogue à un mot dans une langue naturelle.
- **syntax:** les règles qui régissent la structure d'un programme.
- **parsing / analyse:** examiner un programme et analyser la structure syntaxique.
- **bug / bogue:** une erreur dans un programme.
- **débogage:** processus de recherche et de correction de bogues.

1.9 Exercices

Exercice 1

C'est une bonne idée de lire ce livre devant un ordinateur afin que vous puissiez essayer les exemples que vous allez lire.

Chaque fois que vous expérimentez avec une nouvelle fonctionnalité, vous devriez essayer de faire des erreurs. Par exemple, dans le programme «Hello World!», qu'arrive-t-il si vous laissez une des guillemets? Et qu'est-ce qui se passerait si vous quittez les deux? Que se passe-t-il si vous écrivez une mauvaise impression? Ce genre d'expérience vous aide à vous souvenir de ce que vous lisez. Cela vous aide également lorsque vous programmez, parce que vous comprenez ce que signifient les messages d'erreur. Il est préférable de faire des fautes maintenant et que plus tard et accidentellement.

1. Dans un print statement , que se passe-t-il si vous excluez l'une des parenthèses, ou les deux?
2. Si vous essayez d'imprimer une chaîne, que se passe-t-il si vous laissez une des guillemets, ou les deux?
3. Vous pouvez utiliser un signe moins pour créer un nombre négatif comme -2. Que se passe-t-il si vous mettez un + avant un nombre? Qu'en est-il de 2 ++ 2?
4. En notation mathématique, les zéros avancés sont corrects, comme en 02. Que se passe-t-il si vous essayez cela dans Python?
5. Que se passe-t-il si vous avez deux valeurs sans opérateur entre elles?

Exercice 2

Démarrez l'interpréteur Python et utilisez-le comme calculatrice.

1. Combien de secondes existe-t-il en 42 minutes 42 secondes?
2. Combien y a-t-il de miles en 10 kilomètres? Astuce: il y a 1.61 kilomètre dans un mile.
3. Si vous exécutez une course de 10 kilomètres en 42 minutes 42 secondes, quel est votre rythme moyen (temps par mile en minutes et secondes)? Quelle est votre vitesse moyenne en miles par heure?

Chapitre 2
Variables, expressions et déclarations

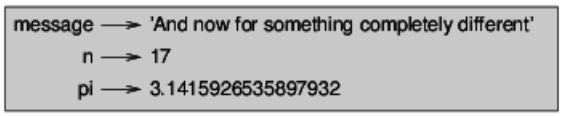
2.1 L'affectation

Une **affectation** crée une nouvelle variable et lui donne une valeur

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.141592653589793
```

Cet exemple fait trois affectations. Le premier attribue une chaîne à une nouvelle variable appelée message; la seconde donne l'entier 17 à n; le troisième attribue la valeur (approximative) de π à pi.

Une manière courante de représenter des variables sur du papier consiste à écrire le nom avec une flèche pointant vers sa valeur. Ce type de figure est appelé diagramme d'état car il indique l'état de chacune des variables (considérez-le comme l'état d'esprit de la variable). La figure 2.1 montre le résultat de l'exemple précédent.



2.2 Nom de variables

Les programmeurs choisissent généralement des noms significatifs pour leurs variables - ils documentent l'utilisation de la variable.

Les noms de variables peuvent être aussi longs que vous le souhaitez. Ils peuvent contenir des lettres et des chiffres, mais ils ne peuvent pas commencer par un chiffre. Il est légal d'utiliser des lettres majuscules, mais il est conventionnel de n'utiliser que des minuscules pour les noms de variables.

Le caractère de soulignement _ peut apparaître dans un nom. Il est souvent utilisé dans les noms comportant plusieurs mots, tels que `your_name` ou `airspeed_of_unladen_swallow` .

Si vous attribuez un nom illégal à une variable, vous obtenez une erreur de syntaxe:

```
>>> 76trombones = 'grand défilé'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
>>> class = 'Zymurgie théorique avancée'
SyntaxError: invalid syntax
```

`76trombones` est illégal car il commence par un nombre. `more@` est illégal car il contient un caractère illégal, @. Mais quel est le problème avec `class`?

Il s'avère que `class` est l'un des mots clés de Python. L'interprète utilise des mots-clés pour reconnaître la structure du programme et ne peut pas être utilisé comme nom de variable.

Python 3 a ces mots-clés:

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

Vous n'êtes pas obligé de mémoriser cette liste. Dans la plupart des environnements de développement, les mots-clés sont affichés dans une couleur différente. si vous essayez d'en utiliser un comme nom de variable, vous le saurez.

2.3 Expression et déclarations

Une expression est une combinaison de valeurs, de variables et d'opérateurs. Une valeur en elle-même est considérée comme une expression, de même qu'une variable. Par conséquent ceux-là sont des expressions juridiques:

```
>>> 42
42
>>> n
17
>>> n + 25
42
```

Lorsque vous tapez une expression au prompt, l'interpréteur l'évalue, ce qui signifie qu'il trouve la valeur de l'expression. Dans cet exemple, `n` a la valeur 17 et `n + 25` a la valeur 42.

Une instruction est une unité de code qui a un effet, comme créer une variable ou afficher une valeur.

```
>>> n = 17
>>> imprimer (n)
```

La première ligne est une instruction d'affectation qui donne une valeur à `n`. La deuxième ligne est une instruction print qui affiche la valeur de `n`.

Lorsque vous tapez une instruction, l'interprète l'exécute, ce qui signifie qu'il fait tout ce que dit l'instruction. En général, les déclarations n'ont pas de valeurs.

2.4 Mode script

Jusqu'à présent, nous avons exécuté Python en mode interactif, ce qui signifie que vous interagissez directement avec l'interpréteur. Le mode interactif est un bon moyen de commencer, mais si vous travaillez avec plus de quelques lignes de code, il peut être maladroit.

L'alternative consiste à enregistrer le code dans un fichier appelé script, puis à exécuter l'interpréteur en mode script pour exécuter le script. Par convention, les scripts Python ont des noms qui se terminent par .py.

Si vous savez créer et exécuter un script sur votre ordinateur, vous êtes prêt. Sinon, je recommande d'utiliser PythonAnywhere à nouveau. J'ai posté des instructions pour l'exécution en mode script à l'adresse http://tinyurl.com/thinkpython2e.

Comme Python fournit les deux modes, vous pouvez tester des bouts de code en mode interactif avant de les insérer dans un script. Mais il existe des différences entre le mode interactif et le mode script qui peuvent prêter à confusion.

Par exemple, si vous utilisez Python comme calculatrice, vous pouvez taper

```
>>> miles = 26,2
>>> milles * 1,61
42.182
```

La première ligne attribue une valeur aux miles, mais elle n'a aucun effet visible. La deuxième ligne est une expression, donc l'interprète l'évalue et affiche le résultat. Il s'avère qu'un marathon est d'environ 42 kilomètres.

Mais si vous tapez le même code dans un script et que vous l'exécutez, vous n'obtenez aucune sortie. En mode script, une expression, par elle-même, n'a aucun effet visible. Python évalue réellement l'expression, mais il n'affiche pas la valeur sauf si vous lui indiquez:

```
miles = 26,2
print(miles * 1,61)
```

Ce comportement peut être déroutant au début.

Un script contient généralement une séquence d'instructions. S'il existe plusieurs instructions, les résultats s'affichent les uns après les autres au fur et à mesure de leur exécution.

Par exemple, le script

```
print(1)
x = 2
print(x)
```

produit la sortie

```
1
2
```

L'instruction d'affectation ne produit aucune sortie.

Pour vérifier votre compréhension, tapez les instructions suivantes dans l'interpréteur Python et voyez ce qu'elles font:

```
5
x = 5
x + 1
```

Maintenant, mettez les mêmes instructions dans un script et exécutez-le. Quelle est la sortie? Modifiez le script en transformant chaque expression en une instruction d'impression, puis réexécutez-la.

2.5 Ordre des opérations

Lorsqu'une expression contient plusieurs opérateurs, l'ordre d'évaluation dépend de l'ordre des opérations. Pour les opérateurs mathématiques, Python respecte les conventions mathématiques. L'acronyme **PEMDAS** est un moyen utile de mémoriser les règles:

- Les **P**arenthèses ont la priorité la plus élevée et peuvent être utilisées pour forcer l'évaluation d'une expression dans l'ordre que vous souhaitez. Comme les expressions entre parenthèses sont évaluées en premier, `2 * (3-1)` est égal à 4 et `(1 + 1) ** (5-2)` est égal à 8. Vous pouvez également utiliser des parenthèses pour faciliter la lecture d'une expression, comme dans `( minute * 100) / 60`, même si cela ne change pas le résultat.
- L'**E**xponentiation a la priorité suivante, donc `1 + 2 ** 3` est égal à 9 et non 27 et `2 * 3 ** 2` à 18 et non 36.
- La **M**ultiplication et la **D**ivision ont une priorité plus élevée que l'addition et la soustraction. Donc, `2 * 3-1` correspond à 5, pas 4, et `6 + 4/2` à 8, pas 5.
- Les opérateurs avec la même priorité sont évalués de gauche à droite (sauf l'exponentiation). Ainsi, dans l'expression `degrees / 2 * pi`, la division se produit en premier et le résultat est multiplié par pi. Pour diviser par 2 π, vous pouvez utiliser des parenthèses ou écrire `degrees / 2 / pi`.

Je ne travaille pas très dur pour me souvenir de la préséance des opérateurs. Si je ne peux pas dire en regardant l'expression, j'utilise des parenthèses pour la rendre évidente.

2.6 Opérations sur les chaînes

En général, vous ne pouvez pas effectuer d'opérations mathématiques sur des chaînes, même si elles ressemblent à des nombres, les opérations suivantes sont donc illégales:

```
'2' - '1'
'oeufs' / 'facile'
'troisième' * 'un charme'
```

Mais il y a deux exceptions, + et * .

L'opérateur + effectue la concaténation de chaînes, ce qui signifie qu'il joint les chaînes en les liant de bout en bout. Par exemple:

```
>>> first = 'throat'
>>> second = 'warbler'
>>> first + second
throatwarbler
```

L'opérateur * travaille également sur les chaînes; il effectue la répétition. Par exemple, `"Spam" * 3` est `"SpamSpamSpam"`. Si l'une des valeurs est une chaîne, l'autre doit être un entier.

Cette utilisation de + et * est logique par analogie avec l'addition et la multiplication. Tout comme `4 * 3` équivaut à `4 + 4 + 4`, nous nous attendons à ce que `'Spam' * 3` soit identique à `'Spam' + 'Spam' + 'Spam'`, et c'est le cas. D'autre part, il existe une différence significative entre la concaténation et la répétition de chaînes et l'addition et la multiplication d'entiers. Pouvez-vous penser à une propriété dont l'addition a que lq concaténation n'ait pas?

2.7 Commentaires

À mesure que les programmes deviennent plus grands et plus compliqués, ils deviennent plus difficiles à lire. Les langages formels sont denses et il est souvent difficile de regarder un morceau de code et de comprendre ce qu'il fait ou pourquoi.

Pour cette raison, il est judicieux d'ajouter des notes à vos programmes pour expliquer en langage naturel ce qu'il fait. Ces notes s'appellent des commentaires et commencent par le symbole `#`:

```
# calcule le pourcentage de l'heure écoulée
pourcentage = (minute * 100) / 60
```

Dans ce cas, le commentaire apparaît sur une ligne à part. Vous pouvez également mettre des commentaires à la fin d'une ligne:

```
pourcentage = (minute * 100) / 60 # pourcentage d'une heure
```

Tout ce qui va du # à la fin de la ligne est ignoré - cela n'a aucun effet sur l'exécution du programme.

Les commentaires sont plus utiles lorsqu'ils documentent des caractéristiques non évidentes du code. Il est raisonnable de supposer que le lecteur peut comprendre ce que fait le code; il est plus utile d'expliquer pourquoi.

Ce commentaire est redondant avec le code et inutile:

```
v = 5 # assigner 5 à v
```

Ce commentaire contient des informations utiles qui ne figurent pas dans le code:

```
v = 5 # vitesse en mètres / seconde.
```

De bons noms de variables peuvent réduire le besoin de commentaires, mais des noms longs peuvent rendre les expressions complexes difficiles à lire, ce qui crée un compromis.

2.8 Débogage

Trois types d'erreur peuvent se produire dans un programme: les erreurs de syntaxe, les erreurs d'exécution et les erreurs sémantiques. Il est utile de les distinguer afin de les localiser plus rapidement.

- **Erreur de syntaxe:** La «syntaxe» fait référence à la structure d'un programme et aux règles relatives à cette structure. Par exemple, les parenthèses doivent être appariées, donc `(1 + 2)` est légal, mais `8)` est une erreur de syntaxe. S'il y a une erreur de syntaxe dans votre programme, Python affiche un message d'erreur et se ferme. Vous ne pourrez pas exécuter le programme. Pendant les premières semaines de votre carrière en programmation, vous passerez peut-être beaucoup de temps à rechercher les erreurs de syntaxe. Au fur et à mesure que vous gagnerez de l'expérience, vous ferez moins d'erreurs et les trouverez plus rapidement.

- **Erreur d'exécution:** Le deuxième type d'erreur est une erreur d'exécution, ainsi appelée car l'erreur n'apparaît pas avant que le programme n'ait commencé à s'exécuter. Ces erreurs sont également appelées exceptions car elles indiquent généralement qu'un événement exceptionnel (et un incident grave) s'est produit. Les erreurs d'exécution sont rares dans les programmes simples que vous verrez dans les premiers chapitres, il faudra donc peut-être un peu de temps avant d'en rencontrer un.

- **Erreur sémantique:** Le troisième type d'erreur est «sémantique», ce qui signifie lié au sens. S'il y a une erreur sémantique dans votre programme, celui-ci s'exécutera sans générer de message d'erreur, mais cela ne fonctionnera pas correctement. Cela fera autre chose. Plus précisément, il fera ce que vous lui avez dit de faire. Identifier les erreurs sémantiques peut être délicat, car cela nécessite de travailler en arrière en examinant les résultats du programme et en essayant de comprendre ce qu'il fait.

2.9 Glossaire

- **variable:** Un nom qui fait référence à une valeur.

- **affectation:** Une déclaration qui assigne une valeur à une variable.

- **diagramme d'état:** Représentation graphique d'un ensemble de variables et des valeurs auxquelles elles se rapportent.

- **mot-clé:** Mot réservé utilisé pour analyser un programme. vous ne pouvez pas utiliser de mots-clés tels que if, def et while comme noms de variable.

- **opérande:** Une des valeurs sur lesquelles opère un opérateur.

- **expression:** Combinaison de variables, d'opérateurs et de valeurs représentant un seul résultat.

- **évaluer:** Simplifier une expression en effectuant les opérations afin de générer une valeur unique.

- **déclaration:** Une section de code qui représente une commande ou une action. Jusqu'à présent, les déclarations que nous avons vues sont des assignations et des déclarations imprimées.

- **exécuter:** Pour exécuter une déclaration et faire ce qu'elle dit.
- **mode interactif:** Une façon d'utiliser l'interpréteur Python en tapant du code à l'invite.
- **mode script:** Une façon d'utiliser l'interpréteur Python pour lire le code d'un script et l'exécuter.
- **scénario:** Un programme stocké dans un fichier.
- **ordre des opérations:** Règles régissant l'ordre dans lequel les expressions impliquant plusieurs opérateurs et opérandes sont évaluées.
- **enchaîner:** Pour joindre deux opérandes de bout en bout.
- **commentaire:** Informations contenues dans un programme destiné à d'autres programmeurs (ou à toute personne lisant le code source) et n'ayant aucun effet sur l'exécution du programme.
- **erreur de syntaxe:** Une erreur dans un programme qui rend impossible l'analyse (et donc impossible à interpréter).
- **exception:** Une erreur détectée pendant l'exécution du programme.
- **sémantique:** Le sens d'un programme.
- **erreur sémantique:** Une erreur dans un programme qui lui fait faire autre chose que ce que le programmeur avait prévu.

2.10 Exercices

Exercice 1

Répétant les conseils du chapitre précédent, chaque fois que vous apprenez une nouvelle fonctionnalité, vous devriez l'essayer en mode interactif et faire des erreurs exprès pour voir ce qui ne va pas.

- *Nous avons vu que* `n = 42` *est légal. Qu'en est-il de* `42 = n` *?*
- *Comment à propos de* `x = y = 1`*?*
- *Dans certaines langues, chaque instruction se termine par un point-virgule„:. Que se passe-t-il si vous mettez un point-virgule à la fin d'une instruction Python?*
- *Que se passe-t-il si vous mettez un point à la fin d'une déclaration?*
- *En notation mathématique, vous pouvez multiplier x et y comme ceci:* `x y`*. Que se passe-t-il si vous essayez cela en Python?*

Exercice 2

Pratique utilisant l'interpréteur Python comme calculatrice:

1. *Le volume d'une sphère de rayon r est 4/3 π r^3. Quel est le volume d'une sphère de rayon 5?*
2. *Supposez que le prix de vente d'un livre est de $24,95 , mais les librairies bénéficient d'une réduction de 40%. La livraison coûte $3 pour le premier exemplaire et 75 cents pour chaque exemplaire supplémentaire. Quel est le prix en gros total pour 60 exemplaires?*
3. *Si je quitte ma maison à 6h52 du matin et que je cours 1 mile à un rythme lent (8:15 par mile), puis 3 miles au tempo (7:12 par mile) et à un mile à nouveau, quelle heure puis-je rentrer à la maison pour le petit déjeuner?*