

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, envoi sur réseau, etc.

Math: 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 ligne de commande, vous n'aurez aucun problème pour installer Python. Mais pour les débutants, il peut être douloureux de connaître le système d'administration 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, dépendent sur 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 est indiquée 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 à bit appelé XOR. Si vous n'êtes pas familiarisé avec les opérateurs bit à 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 (string en anglais), soi-disant parce que les lettres qu'il contient sont enfilées ensemble.

Si vous ne savez pas quel type de valeur a, 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 $2Zz$ n'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.

C'est une phrase anglaise bien structurée avec $t * k$ invalides. Cette phrase valable tokens a, mais structure invalide avec.

Lorsque vous lisez une phrase en anglais 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 tokens, 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 idiom 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 la 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*).

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 "language 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.

1.9 Exercices

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; toutes 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.

Exercice 1.1. 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 1.2. 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?