1 Variables

Les résultats déjà calculés peuvent être gardés en mémoire afin de les réutiliser dans d'autres calculs. Pour cela, on utilise des variables. Les types que nous avons vus jusqu'ici seront plus tard décrits comme *immutables*. Lorsque l'on travaille avec ces types, une variable peut être vue comme une boîte portant un *nom* et contenant une *valeur*. Afin de stocker une valeur dans une boîte, on utilise le symbole d'*affectation* « `=` ».

```
>>> a = 6
```

On accède ensuite à la valeur mémorisée en utilisant le nom de la variable. Avant d'évaluer chaque expression, les noms de variables sont remplacés par les valeurs qu'elles contiennent.

```
>>> a * (a + 1)
 42
```

À gauche du symbole d'affectation `=`, on place le nom de la boîte qui doit être utilisée. À droite, on doit trouver une expression qui sera évaluée en une valeur. Cette valeur sera alors stockée dans la boîte. Une fois qu'une variable est définie, il est possible de la redéfinir avec une nouvelle affectation.

```
>>> a = a + 1
 7
```

Sur cet exemple, le membre de droite est d'abord évalué pour produire la valeur 7. Cette valeur est ensuite stockée dans la variable `a`. L'ancienne valeur est « écrasée » et il n'est plus possible d'y accéder. Ce type d'instruction nous rappelle que le symbole d'affectation est dissymétrique, contrairement au symbole d'égalité utilisé en mathématiques. En particulier, l'instruction « `a + 1 = a` » n'a aucun sens et sera signalée par Python comme une erreur de syntaxe.

Signalons que c'est bien une valeur qui est stockée dans une variable et non une expression. En particulier, si on définit « `b = a * a` » et que l'on change ensuite la valeur de `a`, celle de `b` restera inchangée.

1.0.1 -

Dans cet exercice on s'interdit d'utiliser l'exponentiation « `**` ».

1. Montrer que l'on peut calculer a^8 avec 3 multiplications.
2. Donner deux manières de calculer a^7 avec 4 multiplications.

Python est un langage de programmation à typage dynamique : une même variable peut à un moment donné stocker un entier et plus tard une chaîne de caractères. Le type d'une variable, c'est-à-dire le type de la valeur stockée par cette variable est donc autorisé à changer lors de l'exécution d'un programme. Cette manière de programmer rend cependant les programmes plus difficiles à lire et nous ferons tout pour éviter cela.

Pour les noms de variables, nous nous limiterons aux noms composés de lettres minuscules (`a-z`) et majuscules (`A-Z`), ainsi qu'au caractère « tiret du bas » ou « underscore » (`_`) disponible sur la touche 8 des claviers français. On évitera d'utiliser les accents dans les noms de variables. L'utilisation de chiffres (`0-9`) à la fin d'un nom est autorisée, mais nous éviterons de le faire dans ce cours. Choisir judicieusement le nom de ses variables est un art qu'il est important de cultiver. Les noms de variables courts ont l'avantage d'être rapides à taper et à lire. On les utilisera donc pour stocker des valeurs que nous utiliserons souvent. Les noms de variables longs ont l'avantage d'être plus descriptifs. On les utilisera donc pour faire référence à des valeurs que nous utiliserons plus rarement. Le guide PEP8 recommande de n'utiliser que des minuscules pour les noms de variables. Pour des noms composés de plusieurs mots, il recommande un underscore comme dans `nb_eleves`.

2 État du système

Contrairement aux expressions dont la finalité est de produite une valeur, une affectation a pour effet de changer l'état des variables. On dit qu'elle agit par *effet de bord*. Pour représenter l'état du système, nous utiliserons la notation suivante `{a: 2, b: 7}`. Elle signale que la variable `a` contient la valeur 2 tandis que `b` contient la valeur 7. La programmation *impérative* consiste à écrire une succession d'instructions pour changer l'état de la machine. Le langage machine, qui est le seul compris par les processeurs, fonctionne de cette manière. C'est une des raisons pour lesquelles ce style est central dans de nombreux langages de programmation. C'est le cas pour Python, et c'est un style que nous adopterons souvent ici. Afin de visualiser l'état dans lequel se trouve la machine, on décrira parfois dans ce cours l'état de la machine sur une ligne de commentaire. Ces lignes commencent par le caractère `#` et sont ignorées par Python.

Si par exemple les variables `a` et `b` contiennent respectivement 2 et 7. Les instructions suivantes modifient l'état de la machine comme suit :

```
# etat {a: 2, b: 7}
>>> a = b
    b = a
# etat {a: 7, b: 7}
```

En particulier, ces deux instructions n'ont pas eu pour effet d'échanger le contenu des variables `a` et `b`. La première instruction a eu pour effet d'écraser la valeur contenue dans `a` qui est alors définitivement perdue. Si on possède un verre d'eau et un verre de vin, le meilleur moyen pour échanger le contenu de ces verres est d'utiliser un troisième verre. Pour échanger deux variables, on utilisera donc la séquence d'instructions suivante :

```
# etat {a: 2, b: 7}
>>> c = a
    a = b
    b = c
# etat {a: 7, b: 2, c: 2}
```

Il est possible de supprimer une variable avec l'instruction `del`, mais en pratique, nous nous contenterons d'ignorer les variables dont nous n'avons plus l'utilité.

Dans ce cours, on notera $\mathcal{E}_0, \mathcal{E}_1, \dots$ l'état du système à différentes étapes de l'exécution de notre code. On utilisera souvent la convention suivante : si `a` est une variable, a_k sera la valeur contenue par cette dernière quand le système est dans l'état $\mathcal{E}_k$. Par exemple

```
# etat0 {a: a0, b: b0}
>>> a = a + b
# etat1 {a: a0 + b0, b: b0}
```

signifie qu'après notre affectation $a_1 = a_0 + b_0$ et $b_1 = b_0$.

2.0.1 -

Montrer qu'il est possible d'échanger deux variables contenant des entiers avec des opérations arithmétiques sans utiliser de troisième variable.

3 Entrées, sorties

Le langage Python permet d'interagir avec l'utilisateur en demandant d'entrer des valeurs avec lesquelles il va travailler et en affichant les résultats de son calcul.

Pour afficher des résultats, on utilise la fonction `print`. On l'utilise pour afficher des chaînes de caractères, mais aussi pour afficher des entiers, des nombres flottants ou des nombres complexes.

```
>>> print("Hello world!")
 Hello world!
>>> print(2**10)
 1024
```

L'instruction `print` travaille par effet de bord. Elle a pour effet de changer l'état du système, à savoir ce qui est affiché par la *console* de l'ordinateur. Il est essentiel de bien faire la différence entre l'expression `2**10` qui s'évalue en 1024 et l'instruction `print(2**10)` qui affiche 1024 sur la console. Cette instruction s'évalue en `None`, l'unique valeur du type `NoneType` qui est renvoyée par les fonctions travaillant par effet de bord. La différence peut paraître subtile lorsque l'on travaille avec Python en mode interactif. Dans ce mode, l'instruction `print` est ajoutée automatiquement sans que vous l'ayez demandé.

L'instruction `print` peut être utilisée avec plusieurs valeurs, séparées par des virgules et affichées sur la même ligne les unes à la suite des autres.

```
>>> nb_eleves = 40
>>> print("Il y a", nb_eleves, "élèves dans la classe.")
 Il y a 40 élèves dans la classe.
```

Par défaut, un retour à la ligne est automatiquement inséré après chaque instruction `print`. Pour éviter cela, on peut utiliser l'option `end` et remplacer le caractère « `\n` », par la chaîne de votre choix. Le plus courant est d'utiliser une chaîne vide.

```
>>> print("Hello ", end="")
    print("world!")
 Hello world!
```

Enfin, lorsque vous voulez afficher plusieurs valeurs sur une même ligne en les séparant par des virgules, Python va ajouter un espace entre chaque valeur. Cela peut être utile, mais dans les cas où vous ne le souhaitez pas, vous serez forcés de construire les chaînes de caractères à la main.

```
>>> n = 4
>>> print("Sup" + str(n) + " rocks!")
 Sup4 rocks!
```

L'instruction `input` permet quant à elle de demander des valeurs à l'utilisateur. Quelle que soit la valeur attendue, c'est sous la forme d'une chaîne de caractères que Python la renvoie au programmeur. Il convient donc d'effectuer explicitement une conversion lorsque l'on souhaite une valeur d'un autre type.

```
>>> nom = input("Quel est votre nom ? ")
    entree = input(nom + ", quelle est votre année de naissance ? ")
    annee = int(entree)
    print("Vous aurez", 2024 - annee, "ans l'année de jeux de Paris.")

 Quel est votre nom ? Teddy Riner
 Teddy Riner, quelle est votre année de naissance ? 1989
 Vous aurez 35 ans l'année des jeux de Paris.
```

Bien qu'elle renvoie une valeur, on dit aussi que l'instruction `input` travaille par effet de bord, car elle attend une entrée de l'utilisateur faite au clavier. C'est la seule fois dans ce cours où vous verrez la fonction `input`. Nous sommes en 2019, les téléphones ont des interfaces tactiles et la reconnaissance vocale commence à marcher. Tout cela pour vous dire qu'il vaut mieux laisser gérer l'interface utilisateur par des personnes sachant utiliser ces technologies. Bref, nous vous demandons de ne pas peaufiner l'interface utilisateur de vos programmes. Comme nous travaillerons en mode interactif, `input` sera inutile et nous vous demandons de le laisser de côté.