1 Variables et types

1.1 Assignation de variables

En première approche, une *variable* en Python peut être considérée comme une boîte à laquelle on a donné un nom (son *identifiant*) et contenant une *valeur*. Une instruction qui indique à l'ordinateur de ranger une valeur dans l'une de ces boîtes s'appelle une *assignation de variable*. En Python, elle prend la forme :

```
identifiant = valeur
```

Dans le chapitre précédent, vous avez par exemple déjà rencontré l'instruction :

```
x = 3
```

qui demande d'attribuer la valeur `3` à la variable dont l'identifiant est `x`. Une fois cette instruction traitée par l'interpréteur, ce dernier garde en mémoire le fait que :

— une variable possédant l'identifiant `x` a été définie,

— cet identifiant est rattaché à la valeur `3`.

Dans Spyder, l'*explorateur de variables* liste toutes les variables qui ont été définies jusque là et vous permet de visualiser rapidement leurs identifiants, leurs valeurs, ainsi que d'autres informations. Ne négligez surtout pas cet outil lorsque vous programmerez !

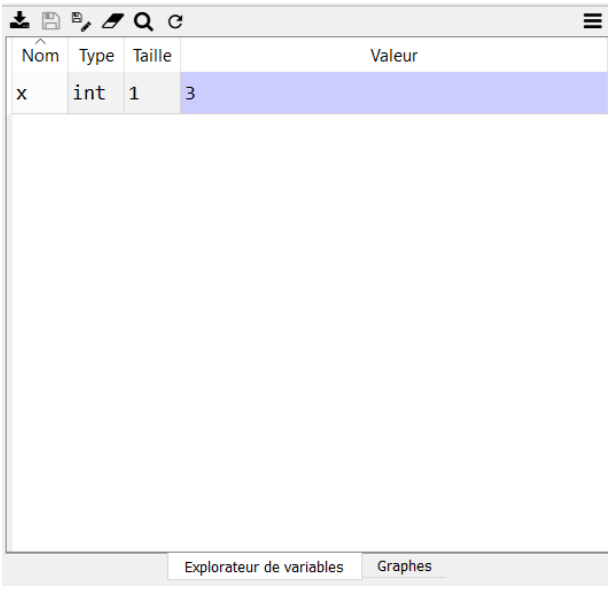

FIGURE 1.1 – L'explorateur de variables de Spyder.

Une fois qu'une variable à été définie à l'aide d'une première assignation, il est possible de :

— l'utiliser dans une *expression*, telle que `3 * x + 2` ,

— refaire une assignation avec le même identifiant, par exemple `x = 5` .

Dans ce dernier cas, la valeur qui était présente dans la variable avant cette nouvelle assignation est définitivement perdue, remplacée par la nouvelle valeur que vous avez donnée. En fait, il est même possible de faire les deux à la fois, avec par exemple l'instruction :

```
x = 3 * x + 1
```

Dans ce cas, l'interpréteur Python commence par calculer l'expression `3 * x + 1` avec la valeur actuellement liée à l'identifiant `x`, et ensuite seulement le résultat obtenu devient la nouvelle valeur de `x`.

À tester. Effectuez chacun des tests suivants dans un interpréteur dédié, afin de bien réinitialiser la mémoire à chaque fois (dans Spyder, fermez toutes les consoles jusqu'à ce qu'une nouvelle console vierge s'ouvre, ou bien ouvrez de vous-même une nouvelle console depuis le menu ≡ du panneau console).

1. Saisissez les instructions suivantes **l'une après l'autre** dans la console Python, tout en observant les changements produits dans l'explorateur de variables de Spyder :

```
x = 4
y = x
x = 2 * x
x = y - x
```

2. Essayez l'instruction ci-dessous **avant** d'avoir assigné une valeur à la variable `x`. Que se passe-t-il?

```
y = x + 3
```

Remarque. Quelques remarques sur le choix de l'identifiant pour une variable.

— L'identifiant peut contenir des lettres, des chiffres ou le cararactère _ (*underscore*). En revanche, il ne peut pas commencer par un chiffre.

— L'identifiant est sensible à la casse, c'est-à-dire que les minuscules et majuscules sont considérées comme des caractères distincts. Par exemple, `BOITE`, `boite` et `BoItE` sont trois identifiants différents.

— Il existe en Python des mots-clés « réservés » qui ne peuvent pas être utilisés comme identifiants pour une variable. Voici la liste de ces mots-clés (elle n'est pas à connaître exhaustivement) :

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

— De manière générale, il est recommandé de choisir des identifiants clairs et concis.

1.2 Types de variables

Plus précisément (vous l'avez peut-être déjà remarqué dans l'explorateur de variables), en Python toute valeur possède un *type*. Le tableau 1.1 recense les principaux types prédéfinis.

À tester. En continuant à jeter un œil à l'explorateur de variables, exécutez le script

```
1 ...
2 print(x)
```

en remplaçant successivement la ligne 1 par les instructions ci-dessous. Que remarquez-vous?

1. `x = 3`
2. `x = 3.0`
3. `x = "3"`

Abr.	Nom complet	Description	Exemple
`int`	*Integer* (Entier)	Représente un nombre entier de manière **exacte**.	`3`
`float`	*Floating-point number* (Nombre flottant*)	Représente un nombre réel de manière **approchée**.	`3.27`
`bool`	*Boolean* (Booléen)	Représente une valeur de vérité. Ne peut prendre que deux valeurs : `True` ou `False`.	`True`
`str`	*String* (Chaîne de caractères)	Représente une suite de caractères (typiquement, du texte).	`"Hello"`

* Ou plus rigoureusement, *nombre en virgule flottante*.

TABLE 1.1 – Principaux types prédéfinis en Python

Attention. On insiste ici sur le fait que le type `float` ne permet **pas** de stocker des réels de manière **exacte** ! Il faut garder en tête qu'il y a toujours une potentielle marge d'erreur entre la valeur exacte qu'on a voulu représenter et la valeur qui est réellement enregistrée en mémoire (voir module IN110). Cela vaut même pour des nombres en apparence « simples ». Pour vous en convaincre, exécutez la commande suivante :

```
print(0.1 + 0.2)
```

Pour chaque abréviation présentée dans le tableau 1.1, une fonction du même nom permet de convertir des valeurs vers ce type (lorsque cela a du sens).

À tester.

1. Dans un interpréteur Python, saisissez l'instruction `x = 2`. Observez ensuite dans cet interpréteur et dans l'explorateur de variables le résultat de chacune des instructions suivantes :
   a) `y = int(x)` b) `y = float(x)` c) `y = bool(x)` d) `y = str(x)`
2. Même question en partant cette fois de l'expression `x = "14"`, puis de l'expression `x = True`.
   Remarque. Vous pouvez retracer l'historique des commandes récentes en utilisant les touches ↑ *et* ↓.
3. Quelle semble être la règle de conversion du type `bool` vers le type `int` ? Et dans le sens contraire ?
4. Que se passe-t-il lorsque la conversion ne fait pas sens, comme dans l'instruction `int("Rien")` ?

1.3 Opérateurs usuels

1.3.1 Opérateurs sur les types numériques

Lorsque les variables `a` et `b` sont toutes deux de type dit « numérique » (c'est-à-dire `int` ou `float`), on peut effectuer les opérations mathématiques usuelles de la façon suivante :

- `a + b` renvoie la somme $a + b$,
- `a - b` renvoie la différence $a - b$,
- `a * b` renvoie le produit $a \times b$,
- `a / b` renvoie le quotient $a \div b$,
- `a ** b` renvoie le nombre a^b.

Si a et b sont de même type, alors les expressions précédentes renvoient un résultat de ce même type, à l'exception de la division entre deux entiers qui renvoie toujours un flottant, même lorsque le quotient est exact [1]. Si les variables a et b sont de types différents (c'est-à-dire l'une est un flottant et l'autre un entier), alors des conversions vers le type flottant sont faites implicitement si nécessaire.

Dans le cas où a et b sont toutes deux de types `int`, on peut de plus effectuer les opérations suivantes :

— `a // b` renvoie le quotient dans la division euclidienne de a par b,

— `a % b` renvoie le reste dans la division euclidienne de a par b.

Remarque. L'opérateur `%` est appelé *modulo*. Il peut, entre autres choses, servir à tester la divisibilité : on rappelle que l'entier a est divisible par b si et seulement si le résultat de `a % b` est zéro. En particulier, a est pair si et seulement si `a % 2` vaut zéro.

Remarque. Pour la culture, sachez qu'il existe aussi les opérateurs `&`, `|` et `^` travaillant *bit à bit* sur la représentation binaire des nombres entiers (qui sera étudiée en IN110).

Exemple	Écriture binaire	Valeur décimale	Interprétation
`a`	$01001101_{(2)}$	77	
`b`	$00011001_{(2)}$	25	
`a & b`	$00001001_{(2)}$	9	« et » bit à bit
`a | b`	$01011101_{(2)}$	93	« ou » bit à bit
`a ^ b`	$01010100_{(2)}$	84	« ou exclusif » bit à bit
`b << 2`	$01100100_{(2)}$	100	Décalage de 2 bits vers la gauche
`b >> 3`	$00000011_{(2)}$	3	Décalage de 3 bits vers la droite

Attention. Il ne faut pas confondre l'opérateur `**` avec l'opérateur `^` qui est parfois utilisé dans d'autres langages pour la mise à la puissance.

À tester.

1. À l'aide de Python, calculer le nombre

$$\frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^5-\left(\frac{1-\sqrt{5}}{2}\right)^5\right).$$

   On peut démontrer que ce nombre vaut exactement 5. Est-ce ce que l'on constate sous Python ?
   Indication. Pour calculer $\sqrt{5}$, on peut se souvenir que $\sqrt{5}=5^{1/2}$.

2. Que donne l'instruction `3 / 0` ? Et l'instruction `3 / (0.1 + 0.2 - 0.3)` ?

Toujours dans le cas où a et b sont de types numériques, on peut de plus effectuer les comparaisons mathématiques usuelles à l'aide des opérateurs décrits dans la table 1.2 : le résultat est alors un booléen.

1. Du moins depuis Python 3. Ça n'était pas le cas en Python 2, où au contraire l'opérateur `/` entre deux entiers donnait toujours un résultat entier (c'était l'équivalent de l'opérateur `//` de Python 3).

Attention, on rappelle que tester une égalité ne fait sens que si a et b sont de type `int`. Entre flottants, le résultat obtenu est à prendre avec précaution en gardant à l'esprit l'existence d'une marge d'erreur.

Opérateur	Interprétation	Équivalent math.
==	« est égal à »	$=$
!=	« est différent de »	$\neq$
<	« est strictement inférieur à »	$<$
>	« est strictement supérieur à »	$>$
<=	« est inférieur ou égal à »	$\leqslant$
>=	« est supérieur ou égal à »	$\geqslant$

TABLE 1.2 – Opérateurs de comparaison entre types numériques

À tester. Commenter les résultats des instructions suivantes :

1. `0.1 + 0.2 <= 0.3`
2. `0.1 + 0.2 == 0.3`
3. `0.1 + 0.2 >= 0.3`
4. `0.1 + 0.2 > 0.3`

1.3.2 Opérateurs sur les booléens

Lorsque les variables a et b sont de types booléen, on peut effectuer les opérations suivantes :

— `a and b` renvoie `True` si a et b valent toutes les deux `True`, et renvoie `False` sinon,

— `a or b` renvoie `True` si au moins l'une des deux variables a ou b vaut `True`, et renvoie `False` sinon,

— `not a` renvoie `True` si a vaut `False`, et renvoie `False` si a vaut `True`.

De plus, on peut comparer les valeurs booléennes contenues dans a et b à l'aide des opérateurs == et !=.

1.3.3 Opérateurs sur les chaînes de caractères

Les chaînes de caractères seront étudiées plus en détail dans un chapitre ultérieur. Toutefois, on peut déjà mentionner le comportement particulier des opérateurs + et * sur ces dernières.

— L'opérateur + sur les chaînes de caractères est un opérateur de *concaténation*, c'est-à-dire qu'il colle bout à bout les chaînes entre elles. Par exemple, l'expression `"Bonjour" + "Toulouse"` donne la chaîne de caractères `"BonjourToulouse"` (notez l'absence d'espace).

— L'opérateur * s'utilise uniquement entre un entier n et une chaîne de caractères S, et sert de notation raccourcie pour `S + S + ... + S` (n fois). Par exemple :

  — `2 * "Cou"` est équivalent à `"Cou" + "Cou"` et donne donc la chaîne `"CouCou"`,

  — `"12" * 3` est équivalent à `"12" + "12" + "12"` et donne donc la chaîne `"121212"`.

On peut également comparer les chaînes de caractères avec les opérateurs présentés dans la table 1.2, et dans ce cas c'est l'ordre alphabétique qui est considéré.

À tester.

1. Analysez le résultat des opérations suivantes :
   a) `5 * "*"` b) `1 * "*"` c) `0 * "*"` d) `-3 * "*"`
2. Analysez le résultat des opérations suivantes :
   a) `"Oui" >= "Non"` b) `"un" < "deux"` c) `"valise" > "valide"`

1.3.4 L'opérateur ternaire

Terminons ce chapitre en introduisant une dernière syntaxe particulière. Il s'agit de l'expression

```
expr1 if expr2 else expr3
```

dans laquelle `expr1` et `expr3` peuvent être des expressions de type quelconque, mais `expr2` doit renvoyer une valeur de type booléen. Cette expression est alors évaluée de la façon suivante :

— si `expr2` vaut `True`, alors l'expression se résume à `expr1`,

— si `expr2` vaut `False`, alors l'expression se résume à `expr3`.

Par exemple :

— l'expression `2 if 0 == 1 else 3` vaut 3,

— l'expression `"A" if 0 != 1 else "B"` vaut `"A"`.

À tester. Déclarer une variable `n` de type entier (initialisée à la valeur de votre choix), puis observez le résultat de l'instruction suivante :

```
print(str(n) + " est un nombre " + ("pair" if n % 2 == 0 else "impair"))
```

Expliquez toutes les opérations en jeu dans cette instruction.

Exercices

Exercice 1.1. Les instructions suivantes déclenchent toutes une erreur :

1) `2.0 * "ABC"`
2) `0.2 ^ 3`
3) `"ABC" ** 2`
4) `"2" ** (1/2)`
5) `int("douze")`
6) `float(True) / int(False)`

Sans utiliser l'ordinateur, associer chacune d'entre elle au message d'erreur qui lui correspond dans la liste ci-dessous. Utiliser ensuite Python pour vérifier vos réponses.

A) `ValueError: invalid literal for int() with base 10: 'douze'`
B) `TypeError: can't multiply sequence by non-int of type 'float'`
C) `TypeError: unsupported operand type(s) for ^: 'float' and 'int'`
D) `TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'`
E) `TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'float'`
F) `ZeroDivisionError: float division by zero`

Exercice 1.2. On suppose déjà définie une variable `n` contenant un nombre entier. Écrire une instruction Python qui affiche la phrase « J'ai mangé n pomme(s). », dans laquelle n sera remplacé par la valeur de la variable `n` (en chiffres), et le mot « pomme » sera accordé correctement (singulier ou pluriel) en fonction de la valeur de n.

Exemples :

— si `n` vaut 1, il faudra que l'instruction affiche « J'ai mangé `1` pomme. »,

— si `n` vaut 3, il faudra que l'instruction affiche « J'ai mangé `3` pommes. ».

Exercice 1.3. On rappelle qu'une année est *bissextile* si elle est divisible par 4 mais pas par 100, à l'exception des années divisibles par 400 qui restent bissextiles. Par exemple :

— l'année 2021 n'est pas bissextile car elle n'est pas divisible par 4,

— l'année 2024 sera bissextile car elle est divisible par 4 mais pas par 100,

— l'année 2000 était bissextile, car divisible par 400,

— l'année 2100 ne sera pas bissextile car elle est divisible par 100 (mais pas par 400).

On suppose déjà définie une variable `annee` de type entier, contenant le numéro d'une année.

1. Écrire une instruction Python qui enregistre dans la variable `bissextile` la valeur `True` si l'année en question est bissextile, et `False` sinon.
   Par exemple, si `annee` *vaut* 2000 *ou* 2024, *la variable* `bissextile` *devra contenir la valeur* `True`, *tandis que si* `annee` *vaut* 2021 *ou* 2100 *alors la variable* `bissextile` *devra contenir la valeur* `False`.
2. Écrire ensuite une instruction qui enregistre dans la variable `phrase` une chaîne de caractère du type « *L'année … est bissextile* » ou « *L'année … n'est pas bissextile* » en fonction de la valeur de la variable `annee`.

À retenir

— Savoir définir une variable, lui affecter une valeur et la modifier.

— Connaître les types de variables principaux en Python, savoir les reconnaître dans une expression.

— Connaître la syntaxe et le fonctionnement des opérateurs sur les types numériques, les booléens et les chaînes de caractères.

— Savoir interpréter et utiliser l'opérateur ternaire.

— Savoir où trouver et comment analyser un message d'erreur basique.

— Fonctions à connaître : `print()`, `int()`, `float()`, `bool()`, `str()`.