version pdf

Chapitre 2 : littéraux et expressions

"Objectifs"

À la fin du chapitre, on doit savoir :

- utiliser la console Python pour évaluer des expressions de littéraux de types `int`, `float`, `str` ou `bool`
- savoir qu'en général on ne peut pas faire d'opération entre deux valeurs de types différents
- comprendre un message d'erreur `ZeroDivisionError` ou `TypeError`

Éditeur et console Python

"Définition 1"

Dans un environnement de programmation Python, appelé aussi **IDE**, comme Spyder ou Thonny, on distingue souvent deux espaces :

- **l'éditeur** : il permet d'écrire un programme dans un fichier texte, généralement avec l'extension `.py` ;
- **l'interpréteur interactif**, ou **console Python** : il permet de tester rapidement des expressions ou des instructions.

Dans la console Python, l'invite ou prompt `>>>` indique que Python attend une saisie. Lorsqu'on écrit une expression correcte, Python l'évalue et affiche sa valeur.

"Exemple 1"

```
>>> 2 + 3
    5
>>> "Bon" + "jour"
'Bonjour'
```

"Attention"

La console est pratique pour faire des essais rapides.

Pour écrire un programme que l'on veut onserver, on utilise plutôt l'éditeur et on enregistre le fichier avec l'extension `.py`.

Littéraux et types

"Définition 2"

En Python, un **littéral** est une valeur écrite directement dans le code source.

Chaque littéral possède un **type**, c'est-à-dire une catégorie qui détermine les opérations possibles sur cette valeur.

Il existe quatre types de base importants :

Type	Signification	Exemples de littéraux
`int`	nombre entier	`0`, `42`, `-7`
`float`	nombre décimal approché	`3.14`, `-0.5`, `2.0`
`bool`	valeur logique	`True`, `False`
`str`	chaîne de caractères	`"NSI"`, `'Bonjour'`, `"42"`

"Attention"

Le littéral `42` est un entier de type `int`, alors que le littéral `"42"` est une chaîne de caractères de type `str`.

"Observer le type d'une valeur"

La fonction `type` permet d'obtenir le type d'une valeur.

```
>>> type(42)
<class 'int'>
>>> type(3.14)
<class 'float'>
>>> type("42")
<class 'str'>
>>> type(True)
<class 'bool'>
```

"Conversion de type"

Dans certains cas, on peut changer le type de certaines valeurs en appliquant le constructeur du type cible. La possibilité dépend du type de départ et du type cible. On peut convertir n'importe quelle valeur en chaîne de caractère avec `str(valeur)` ou en booléen avec `bool(valeur)` mais on ne peut pas convertir n'importe quelle valeur en entier ou flottant.

```
>>> str(2) # conversion de int vers str
'2'
>>> float('1.5') # conversion de str vers float
1.5
>>> int('01')
ValueError
```

"Exercice 1"

Pour chaque littéral, indiquer son type.

Littéral	Type
`17`	....
`-4`	....
`2.5`	....
`"Bonjour"`	....
`'3.14'`	....

Littéral	Type
`True`	....
`"False"`	....
`0.0`	....

Expressions

"Définition 3"

Une **expression** est un morceau de code constituée d'un littéral ou d'opérations sur des littéraux ou des variables, que Python peut évaluer pour produire une valeur.

Une expression possède donc une **valeur** et un **type**.

"Exemple 2"

```
>>> 2 + 3
5
>>> type(2 + 3)
<class 'int'>
>>> 2 + 3.5
5.5
>>> type(2 + 3.5)
<class 'float'>
```

Opérateurs sur les entiers

"Opérateurs sur les entiers"

Fonction de l'opérateur	Syntaxe	Exemple
addition	`+`	`7 + 3` donne `10`
soustraction	`-`	`7 - 3` donne `4`

Fonction de l'opérateur	Syntaxe	Exemple
multiplication	`*`	`7 * 3` donne `21`
puissance	`**`	`7 ** 3` donne `343`
quotient de la division euclidienne	`//`	`17 // 3` donne `5`
reste de la division euclidienne	`%`	`17 % 3` donne `2`

"Exercice 2"

Évaluer les expressions suivantes, puis vérifier dans la console Python.

Expression	Valeur
`10 + 4`	....
`10 - 4`	....
`10 * 4`	....
`10 ** 4`	....
`14 // 4`	....
`14 % 4`	....

Opérateurs sur les flottants

"Opérateurs sur les flottants"

Fonction de l'opérateur	Syntaxe	Exemple
addition	`+`	`1.5 + 2.0` donne `3.5`
soustraction	`-`	`1.5 - 2.0` donne `-0.5`
multiplication	`*`	`1.5 * 2.0` donne `3.0`
division	`/`	`3.0 / 2.0` donne `1.5`
puissance	`**`	`2.0 ** 3` donne `8.0`

"Nombres décimaux"

Les valeurs de type `float` sont des nombres décimaux représentés de façon approchée par l'ordinateur.

Il peut donc arriver que certains calculs donnent un résultat avec de petites imprécisions d'affichage.

```
>>> 0.1 + 0.2
0.30000000000000004
```

"Exercice 3"

Évaluer les expressions suivantes, puis vérifier dans la console Python.

Expression	Valeur
`1.5 + 2.5`	....
`4.0 - 1.5`	....
`2.5 * 4`	....
`5.0 / 2.0`	....
`2.0 ** 4`	....
`7.0 + 0.5`	....
`type(7.0 + 1.0)`	....

Opérateurs sur les chaînes de caractères

"Opérateurs sur les chaînes de caractères"

Fonction de l'opérateur	Syntaxe	Exemple
concaténation	`+`	`"Bon" + "jour"` donne `"Bonjour"`
répétition	`*`	`"ha" * 3` donne `"hahaha"`

Fonction de l'opérateur	Syntaxe	Exemple
longueur	`len(...)`	`len("NSI")` donne `3`

"Définition"

La **concaténation** consiste à coller plusieurs chaînes de caractères les unes à la suite des autres.

"Attention"

L'expression `"2" + "3"` ne fait pas une addition : elle colle deux chaînes de caractères.

```
>>> "2" + "3"
'23'
```

"Exercice 4"

Évaluer les expressions suivantes, puis vérifier dans la console Python.

Expression	Valeur
`"a" + "b"`	....
`"Bon" + "jour"`	....
`"ha" * 4`	....
`3 * "go"`	....
`"2" + "3"`	....
`len("Python")`	....
`type("42")`	....

Opérateurs sur les booléens

"Définition 4"

Une valeur de type `bool` est une valeur logique.

Son nom est un hommage à George Boole qui est le premier mathématicien à formaliser des calculs logiques au dix neuvième siècle

Il n'existe que deux valeurs booléennes : `True` et `False`.

"Comparaisons"

Les comparaisons produisent des valeurs booléennes.

Sens de l'opérateur	Syntaxe	Exemple
égal à	`==`	`3 == 3` donne `True`
différent de	`!=`	`3 != 4` donne `True`
strictement inférieur	`<`	`3 < 4` donne `True`
inférieur ou égal	`<=`	`3 <= 3` donne `True`
strictement supérieur	`>`	`5 > 8` donne `False`
supérieur ou égal	`>=`	`5 >= 2` donne `True`

"Attention"

En Python, `=` et `==` n'ont pas le même rôle.

- `==` sert à comparer deux valeurs.
- `=` servira plus tard à affecter une valeur à une variable.

"Propriété 1 : Opérateurs booléens de base"

Si un opérateur booléen a n opérandes, comme chacun ne peut prendre que 2 valeurs, cela donne un nombre fini de 2^n possibilités pour les valeurs des opérandes. Pour chaque opérateur, on résume les résultats pour ces 2^n cas dans une **table de vérité**.

"Opérateur `not` (négation logique)"

$n = 1$ opérande donc $2^1 = 2$ cas dans la table de vérité.

`a`	`not a`
`False`	`True`
`True`	`False`

"Opérateur `or` (ou logique)"

$n = 2$ opérandes donc $2^2 = 4$ cas dans la table de vérité.

`a`	`b`	`a or b`
`False`	`False`	`False`
`False`	`True`	`True`
`True`	`False`	`True`
`True`	`True`	`True`

"Opérateur `and` (et logique)`"

$n = 2$ opérandes donc $2^2 = 4$ cas dans la table de vérité.

`a`	`b`	`a and b`
`False`	`False`	`False`
`False`	`True`	`False`
`True`	`False`	`False`
`True`	`True`	`True`

"Exercice 5"

Évaluer les expressions suivantes, puis vérifier dans la console Python.

Expression	Valeur
`3 < 5`	....
`3 == 5`	....
`3 != 5`	....
`not True`	....
`True and False`	....
`True or False`	....
`(3 < 5) and (10 == 10)`	....
`(3 > 5) or (10 == 10)`	....

Erreurs

"Erreur de type `ValueError`"

L'erreur `ValueError` apparaît lorsqu'une fonction reçoit une valeur du bon type mais d'une valeur impossible à utiliser.

Par exemple, la fonction `int` peut convertir une chaîne de caractères en entier uniquemebt si cette chaîne représente bien un entier.

```
>>> int("42")
42
>>> int("4.2")
ValueError: invalid literal for int() with base 10: '4.2'
```

"Erreur de type `ZeroDivisionError`"

L'erreur `ZeroDivisionError` apparaît lorsqu'on tente de diviser par zéro.

```
>>> 5 / 0
ZeroDivisionError: division by zero

>>> 5 // 0
ZeroDivisionError: integer division or modulo by zero

>>> 5.0 / 0.0
ZeroDivisionError: float division by zero
```

"Erreur de type `TypeError`"

L'erreur `TypeError` apparaît lorsqu'une opération est impossible avec les types utilisés.

Par exemple, Python ne peut pas additionner directement une chaîne de caractères et un entier :

```
>>> "2" + 3
TypeError: can only concatenate str (not int) to str
```

"Conversion de type implicite"

Python peut parfois convertir automatiquement une valeur d'un type vers un autre type. C'est le cas entre `int` et `float`.

```
>>> 2 + 3.5
5.5
>>> type(2 + 3.5)
<class 'float'>
```

En revanche, Python ne convertit pas automatiquement une chaîne de caractères en nombre.

"Exercice 6"

Pour chaque expression, indiquer si elle produit une valeur ou une erreur.

En cas d'erreur, préciser s'il s'agit de `TypeError`, `ZeroDivisionError` ou `ValueError`.

Expression	Valeur ou erreur
`8 / 2`	....
`8 / 0`	....
`8 // 0`	....
`"a" + "b"`	....
`"a" + 3`	....
`"a" * 3`	....
`"a" - "b"`	....
`2 + 3.5`	....
`int('bonjour')`	....