1 - Expressions

qkzk

2024/06/29

pdf: pour impression

Symboles utilisés pour présenter du code

On distingue deux types d'exécution en Python

Lorsqu'on exécute tout un fichier avec :

```
python mon_script.py
```

et lorsqu'on exécute Python en mode intéractif :

```
python
```

Dans le second cas, on arrive devant l'*interpréteur*. On le sait parce qu'il présente chaque ligne avec trois chevrons :

```
>>>
```

Le code que vous lirez sera soit tiré d'un fichier python (.py) soit de l'interpréteur.

- Dans le premier cas, le code est écrit directement.

  ```
  a = 2
  print(a)
  ```

- Dans le second les instructions sont précédées de trois chevrons

  ```
  >>> a = 2
  >>> a
  2
  ```

  L'interpéteur affiche le valeur de la dernière expression.

  Le nombre `2` qu'on voit apparaître n'a pas été tapé à la main mais est la valeur de la variable 2.

Expressions

Python exécute les instructions d'un script python (dont l'extension est `.py`) de haut en bas.

Chaque ligne de code est lue, traduite par l'interpréteur et exécutée.

```
3 + 4
4 - 9
0.12 * 2
```

Chaque ligne précédente est une *expression*, le résultat de l'opération est la *valeur* de cette expression.

Ainsi les valeurs des expressions précédentes sont `7`, `-5` et `0.24`.

Commentaires

Tout ce qui suit un symbole `#` (dièse) n'est pas interprété par Python.

C'est un commentaire.

Les commentaires ont plusieurs usages :

- permettre d'informer le lecteur sur le déroulement d'un programme,
- rendre un morceau de code inopérant le temps de sa correction.

```
# 34 == 4
34
23 == 12 + 11   # je crois que c'est vrai !
```

La première ligne de code ci-dessus est un commentaire, comme la phrase : "`je crois que c'est vrai !`".

Les expressions ont toute un *type* sur lequel nous reviendrons plus tard.

Opérations mathématiques courantes

Opération	symbole	exemple	résultat
addition	`+`	`2 + 5`	`7`
soustraction	`-`	`5 - 6`	`-1`
multiplication	`*`	`3.1 * 2`	`6.2`
division flottante	`/`	`4 / 3`	`1.333...`
division entière	`//`	`5 // 3`	`1`
reste (*modulo*)	`%`	`5 % 3`	`2`
puissance	`**`	`3 ** 2`	`9`

Types

C'est un sujet délicat auquel nous consacrerons plusieurs chapitres mais déjà qu'est-ce qu'un *type* ?

> En programmation informatique, un **type de donnée**, ou simplement un **type**, définit la nature des valeurs que peut prendre une donnée, ainsi que les opérateurs qui peuvent lui être appliqués.

Donc c'est un type c'est :

1. un ensemble de valeurs possibles,
2. un ensemble d'opérations possibles sur ces valeurs.

Par exemple `1` est du type `int`, on peut réaliser certaines opérations *arithmétiques* sur ce type (addition etc.)

En Python, tous les objets ont un type.

Types élémentaires

Type	Description	Exemple
`int`	Entiers positifs ou négatifs	`0`, `1234`, `-32`
`float`	Nombres à virgule	`0.234`, `-43.483`
`bool`	Booléens (Vrai ou Faux)	`True`, `False` (seules valeurs)
`NoteType`	Rien, l'absence de réponse...	`None` (seule valeur)
`str`	Du texte	`'Super Content'`, `"James Bond"`, `"123.34"`

On accède au type d'une expression avec la fonction `type` :

```
>>> type("Super Kevin")
<class 'str'>
>>> type(123)
<class 'int'>
>>> type("123")  # attention aux guillemets !
<class 'str'>
```

Nous travaillerons avec d'autres types mais une chose à la fois.

Exercices

Exercice 1 - évaluer des expressions Évaluer de tête les expressions suivantes, *ensuite* vérifiez les dans l'interpréteur

```
34 // 3
34 % 3
34 / 3
4 ** 2
4 ** 0.5
9 ** 0.5
34 == 13
34 == 31 + 3
```

Quelle est la priorité opératoire de l'opérateur `==` et de `+` ?

Exercice 2 - lire un message d'erreur

1. Créer un fichier python (.py) dans Thonny contenant le code suivant :

```
3 / 0
```

2. Exécutez le. Vous devriez obtenir la sortie suivante :

```
python zd.py
Traceback (most recent call last):
  File "/home/quentin/zd.py", line 1, in <module>
    3 / 0
ZeroDivisionError: division by zero
```

Ce message d'erreur se lit facilement de *bas en haut* :

1. Le script provoque une erreur.
2. Elle est du type `ZeroDivisionError` (on s'en doutait !)
3. Elle est située dans le module `zd.py` à la ligne 1.

Ce qui permet facilement de la repérer.

Les erreurs courantes sont :

- `TypeError` : une opération non définie exemple : `3 + "Super"`,
- `ZeroDivisionError` : division par zéro,
- `ValueError` : erreur générique lorsqu'une expression est du bon type mais pas de la bonne valeur.
- `IndexError` : erreur d'indice pour une `list`
- `KeyError` : erreur avec une clé de dictionnaire. etc.

Exercice 3 - affichage

Lorsqu'on exécute des instructions dans l'interpréteur, la dernière valeur est affichée.

Lorsqu'on exécute un script, rien n'est affiché sauf si on le demande.

C'est le rôle de la fonction `print`. Elle permet d'afficher dans la console la valeur d'une expression.

```
print("Quentin")
print(1 + 2 == 5)
```

Va produire l'affichage :

```
Quentin
False
```

Ce sont bien les valeurs des expressions demandées.

Comprenez bien qu'un affichage ne sert qu'à l'utilisateur...

- L'expression `1 + 2 == 5` vaut `False`
- L'expression `print(1 + 2 == 5)` vaut `None`

1. Vérifier ces deux affirmations

`print` accepte un nombre quelconque de paramètres d'entrée, séparés par des virgules :

```
>>> print("Hello", "Harry")
Hello Harry
>>> print("NSI", "number", 3 - 2)
NSI number 1
```

Lors de l'affichage, ils sont d'abord évalués puis séparés par des espaces.

2. Produire l'affichage suivant, en remplaçant par vos noms et prénoms :

```
Mon nom est Bond, James Bond.
```

Affecter :

attribuer une valeur à une variable.

Variable :

symbole qui associe un nom à une valeur.

Affecter, c'est donc donner un *nom* à une *valeur*.

Ainsi qu'on l'a déjà vu, il est possible de changer la valeur associée à ce nom.

```
prenom = "Raoul"
prenom = "Jean-Kev"
print(prenom)
```

Va afficher : `Jean-Kev`, la dernière valeur associée à `prenom`.

{{< python >}}prenom = "Raoul" prenom = "Jean-Kev" print(prenom) {{< /python >}}

Ajoutez une ligne `print(prenom)` entre les affectations si n'êtes pas sûr d'avoir compris.

Exercice 3 - affectation

1. Créer une variable `somme` contenant la somme des entiers de 1 à 10.
2. Créer une copie de cette variable `deuxieme_somme` et vérifier que `somme == deuxieme_somme`.
3. Modifier la valeur de `somme` en lui ajoutant 11.
4. A-t-on l'égalité entre `somme` et `deuxeme_somme` après cette étape ?
5. Vérifier l'état des variables dans Thonny ou Pythontutor.
6. Afficher la taille en octets puis en bits d'un fichier de 536ko. *On donne : 1ko (1 kilooctet) = 1000 octets et 1 octet = 8 bits.*

Types d'une variable

En Python, toutes les valeurs ont un type. `12` est du type `int`, `True` est du tyle `bool` etc.

On verra souvent "*le type de la variable*", on devrait dire : *type de la valeur associée à la variable* ou *type de la valeur de l'expression*, c'est un abus de langage.

Exercice 4 - reconnaître un type

Déterminer *de tête* le type des variables suivantes puis vérifier dans l'interpréteur.

```
age = 121
poids = 256.56
taille = 30 / 3
pays = "France"
ville = "bour" * 2 + "g"
```

Les variables python peuvent changer de valeur durant l'exécution d'un programme (on dit qu'elles sont *mutables*) elles peuvent aussi changer de type (le typage est dit *dynamique*). C'est impossible dans beaucoup d'autres langages.

```
a = 123
a = "SUPER"
a = 12 / 13
```

Le code précédent est valide mais est *horrible*. On évitera au maximum de changer le type d'une variable durant l'exécution d'un programme.

Exercice 5

Affichez les types des variables après chaque affectation :

```
a = 123
a = "SUPER"
a = 12 / 13
```

Instruction et expression

- Une *epression* est un calcul qui a une *valeur*.
- Une *instruction* est un ordre qui change l'état de la machine.

Dans l'exemple ci-dessous :

```
a = 3 * 3 + 2
2 * a + 5
print(4 * a)
```

1. La première ligne affecte une valeur à `a`, c'est une *instruction*. Elle contient une *expression* : `3 * 3 + 2`.
2. La seconde ligne calcule une valeur et n'en fait rien. `2 * a + 5` est une *expression*.
3. La troisième ligne affiche la valeur de l'*expression* `4 * a`, c'est une *instruction*.