

Introduction à la programmation avec Python

September 6, 2024

1 Arithmétique, variables, instructions

1.1 Les opérateurs arithmétiques

Priorité opératoire

En Python, on peut saisir des combinaisons arbitraires d'**opérations arithmétiques**, à l'aide des quatre opérations les plus communes : + ; - ; * ; / et de l'opération « élever à la puissance » **.

► Code à tester :

```
2**2 + 5 * (10 - 1 / 2)
```

```
[ ]: # Code à tester
```

Remarque : les espaces ne modifient pas la façon dont l'expression est interprétée.

Les expressions $1+2 * 3$ et $1 + 2*3$ sont équivalentes pour l'interprète Python.

Error:

- L'interprète n'accepte que des expressions arithmétiques complètes et bien formées. Sinon, l'interprète stop l'exécution du code et indique la présence d'une erreur. Le code ci-dessous génère une **erreur de syntaxe**, c'est à dire une instruction mal formée.

► Code à tester :

```
1 + * 2
```

- Il existe un autre type d'erreur lorsque l'on donne à l'interprète une expression qui est correcte du point de vue de l'écriture mais dont le résultats ne correspond pas à ce que l'on veut obtenir, c'est une **erreur sémantique**. Aucun message d'erreur n'étant affiché, ces erreurs sont donc plus difficiles à identifier lors du débogage du code.

► Code à tester :

```
2024 + 2 # je veux obtenir le double de 2024
```

- Attention, les nombres « à virgule » s'écrivent avec la norme anglo-saxonne en utilisant un point comme séparateur décimal. La virgule a un autre sens en Python, que l'on abordera plus tard.

► Code à tester :

```
1,2 + 3,4
1,2 * 3
```

```
[ ]: # Code à tester
```

Division euclidienne

De nombreux programmes manipulent des entiers et nécessitent l'utilisation de la division euclidienne avec :

- L'opérateur « *quotient dans la division entière par* » `//` ;
- L'opérateur « *reste dans la division entière par* », aussi appelé « *modulo* », `%` .

► **Code à tester** : Compléte le tableau suivant en remplaçant a et b par les valeurs indiquées dans chacun des cas.

	$a=7$ et $b=3$	$a=-7$ et $b=3$	$a=7$ et $b=-3$
$a // b$
$a \% b$

Question : Le théorème de la division euclidienne (énoncé ci-dessous) est-il vérifié dans tous les cas?

```
[ ]: # Code à tester
```

Théorème de la division euclidienne dans les entiers relatifs

Soit a et b deux entiers relatifs, $b \neq 0$.

Il existe un unique *entier relatif* q (le quotient) et un unique *entier naturel* r (le reste) tel que :

$$a = b \times q + r \quad \text{et} \quad 0 \leq r < |b|$$

1.2 Variables

Instruction d'affectation

Les résultats calculés peuvent être mémorisés par l'interprète, afin d'être réutilisés plus tard dans d'autre calcul.

► **Code à tester** :

```
a = 1 + 1
```

```
[ ]: # Code à tester
```

La notation `a =` permet de donner un nom à la valeur à mémoriser. L'interprète effectue l'opération `1 + 1` et le mémorise dans la variable `a`. Aucun résultat n'est affiché. On accède à la valeur mémorisée en utilisant le nom `a`.

► **Code à tester :**

```
a
```

```
[ ]: # Code à tester
```

Plus généralement, la variable peut être utilisée dans la suite des calculs.

► **Code à tester :**

```
a * (1 + a)
```

```
[ ]: # Code à tester
```

Le symbole `=` utilisé pour introduire la variable `a` désigne une opération **affectation**. Il attend à sa gauche un nom de variable et à sa droite une expression. On peut donner une nouvelle valeur à la variable `a` avec une nouvelle affectation. Cette valeur remplace la précédente.

► **Code à tester :**

```
a = 3
```

```
a * (1 + a)
```

```
[ ]: # Code à tester
```

Le calcul de la nouvelle valeur de `a` peut utiliser la valeur courante de `a`.

► **Code à tester :**

```
a = a + 1
```

```
a
```

```
[ ]: # Code à tester
```

Un nom de variable peut être formé de plusieurs caractères (lettres, chiffres et tiret de soulignement). Il est recommandé de ne pas utiliser de caractères accentués et l'usage veut que l'on se limite aux caractères minuscules.

► **Code à tester :**

```
cube = a * a * a
```

```
ma_variable = 48
```

```
ma_variable2 = 2024
```

```
[ ]: # Code à tester
```

Error:

- Un nom de variable ne doit pas commencer par un chiffre et certains noms sont interdits (mots réservés du langage)

► **Code à tester :**

```
4x = 2
def = 3
```

- Il n'est pas possible d'utiliser une variable qui n'a pas encore été définie.

► **Code à tester :**

```
b + 1
```

- Seul un nom de variable peut se trouver à la gauche d'une affectation.

► **Code à tester :**

```
1 = 2
```

```
[ ]: # Code à tester
```

Représentation d'une variable

Une variable peut être imaginé comme une petite boîte portant un *nom* (ou une étiquette) et contenant une *valeur*.

Une variable déclarée avec $x = 1$ peut se représenter par une boîte appelée x contenant la valeur 1.

x 1

Lorsque l'on modifie la valeur de la variable x , par exemple avec l'affectation $x = x + 1$, la valeur 1 est remplacée par la valeur 2.

x 2

État

- À chaque étape de l'exécution des instructions par l'interprète, chacune des variables introduites contient une valeur. L'ensemble des associations entre des nom de variables et des valeurs constitue l'**état** de l'interprète Python.

À l'issue de l'exécution des deux instructions suivantes,

```
a = 2
b = 3
```

l'état de l'interprète peut être représenté par l'ensemble :

$\{ a \text{ 2 } ; b \text{ 3 } \}$

On obtient après exécution de l'instruction

```
a = a + b
```

un nouvel état de l'interprète, où seule la valeur de a a été modifié

$\{ a \text{ 5 } ; b \text{ 3 } \}$

- Lorsqu'une variable est affectée pour la première fois, ou **initialisée**, l'association correspondante apparaît dans l'état. Après l'exécution de

```
c = a + b
```

on obtiendra l'état

$$\{ a \boxed{5}; b \boxed{3}; c \boxed{8} \}$$

- Lorsqu'on affecte une nouvelle variable avec la valeur d'une variable déjà existante, comme

```
d = a
```

il est important de comprendre que **a** et **d** ne sont pas deux noms pour la même boîte, mais bien deux boîtes différentes.

$$\{ a \boxed{5}; b \boxed{3}; c \boxed{8}; d \boxed{5} \}$$

La modification de la valeur de **a** n'aura aucun effet sur celle de **d**.

```
a = 7
```

$$\{ a \boxed{7}; b \boxed{3}; c \boxed{8}; d \boxed{5} \}$$

- Pour mettre en évidence l'indépendance des variables, on peut utiliser l'instruction

```
id(a)
```

qui permet d'obtenir l'identifiant unique d'une variable (le numéro unique attribué à chaque boîte), qui est usuellement son adresse dans la

mémoire de la machine exécutant le code.

► Code à tester :

Exécute toutes les instructions ci-dessus, puis affiche tous les identifiants des variables déclarées avec l'instruction `id`.

```
[ ]: # Code à tester
```

2 Les instructions d'entrée/sortie

2.1 Affichage

Pour afficher les résultats d'une expression calculée ou le contenu d'une variable, on utilise une **instruction de sortie** qui s'appelle `print`.

► Code à tester:

```
print(1+2)
a = 2.3
print(a)
```

```
[ ]: # Code à tester
```

On peut aussi afficher un message, écrit entre guillemets.

► **Code à tester :**

```
print("Hello, World!")
```

Le texte écrit entre guillemets est appelé une **chaîne de caractères**. Il s'agit d'un type de base, tout comme les entiers et les flottants. Tous les caractères sont autorisés (accentués, spéciaux, smileys...) puisque cette chaîne n'est pas interprétée par Python. Pour s'en convaincre il suffit d'observer la différence entre ces deux instructions :

► **Code à tester :**

```
print(1+2)
print("1+2")
```

```
[ ]: # Code à tester
```

On peut également faire afficher plusieurs éléments à la fois, entre les séparants par une virgule.

► **Code à tester :**

```
a=1
b=2
print("la différence de",a,"avec",b,"vaut",a-b)
```

2.2 Intéragir avec l'utilisateur

Pour permettre l'interaction du programme *en cours d'exécution* avec l'utilisateur, par exemple la saisie de la valeur d'une variable entière, il faut procéder en deux temps :

- on utilise d'abord l'instruction **input** pour récupérer tous les caractères tapés sur le clavier, sous la forme d'une *chaîne de caractères*,
- puis on convertit cette chaîne de caractère en un nombre entier.

```
saisie = input()
a = int(saisie)
print("le nombre suivant est", a+1)
```

```
[ ]: # Code à tester
```

L'instruction **input** est une **instruction d'entrée** qui interrompt l'exécution du programme et attend que l'utilisateur saisisse des caractères au clavier. La saisie se termine lorsqu'il appuie sur la touche .

Error:

- Les parenthèses font parties de la syntaxe des instructions d'entrée et de sortie. En cas d'oubli on obtient une erreur de syntaxe `SyntaxError` :

```
print 3
```

- Les guillemets utilisés avec l'instruction `print` doivent obligatoirement être fermés pour ne pas obtenir là encore une erreur de syntaxe `SyntaxError`.

```
print("3")
```

- Pour manipuler les caractères saisis comme des nombres, il faut obligatoirement les convertir, sans quoi on obtient une erreur d'exécution `TypeError` ; on ne peut pas ajouter à un nombre à une chaîne de caractères tout comme en mathématiques on n'ajoute pas un nombre à une droite.

```
1 + "2"
```