Arithmétique, variables, instructions

Cours

1. Un peu d'histoire de python

Python est un langage de programmation dit *interprété*. Il a été développé en 1989 par Guido Von Russum, à l'université d'Amsterdam.

Vocabulaire à connaître :

Un langage interprété (contrairement à un langage compilé) est retranscrit en langage machine au fur et à mesure de son exécution via un interprète.

Le langage machine, ou code machine, est la suite de bits qui est interprétée par le processeur d'un ordinateur exécutant un programme informatique. C'est le langage natif d'un processeur, c'est-à-dire le seul qu'il puisse traiter. Il est composé d'instructions et de données à traiter codées en binaire.

Python est devenu un langage de programmation généraliste très utilisé autant dans l'industrie que dans le milieu académique et scolaire.

Python est un langage Open Source (libre de droit et gratuit) développé et utilisé par une large communauté d'utilisateurs.

Un logiciel Open Source est un code conçu pour être accessible au public : n'importe qui peut voir, modifier et distribuer le code à sa convenance.

Ex : Le navigateur web Mozilla Firefox, le lecteur multimédia VLC media player ...

2. Mode interactif

Le langage de programmation python permet d'interagir avec la machine à l'aide d'un programme appelé interprète python.

On peut l'utiliser de 2 façons différentes :

- Méthode 1 : consiste en un dialogue avec l'interprète. **C'est le mode interactif.**
- Méthode 2 : écrire un programme dans un fichier source puis le faire exécuter par l'interprète. **C'est le mode programme.**

Les trois chevrons >>> constitue l'invite de commandes de python, qui indique qu'il attend des instructions. Si par exemple, on tape 1+2 puis la touche entrée, l'interprète Python calcule et affiche le résultat.

```
>>> 1+2
3
>>>
```

Comme on le voit, les chevrons apparaissent de nouveau. L'interprète est prêt à recevoir de nouvelles instructions.

Arithmétique

En Python, on peut saisir des combinaisons arbitraires d'opérations arithmétiques, en utilisant notamment les quatre opérations les plus communes.

```
>>> 2+5*(10-1/2)
49.5
>>>
```

L'addition est notée avec le symbole +, la soustraction avec le symbole -, la multiplication avec le symbole * et la division avec le symbole /. La priorité des opérations est usuelle et on peut utiliser des parenthèses.

```
>>> 1+2 * 3
7
```

Erreurs

L'interprète n'accepte que des expressions arithmétiques complètes et bien formées. Sinon, l'interprète indique la présence d'une erreur.

```
>>> 1 + * 2
```

Ici, vous obtiendrez le message **SyntaxError : invalid syntax**. Cela indique une erreur de syntaxe, c'est-à-dire une instruction mal formée.

Les nombres et Python

Les nombres de Python sont soit des entiers relatifs, simplement appelés **entiers**, soit des nombres décimaux, appelés **flottants**.

Les entiers peuvent être de taille arbitraire.

Ces entiers ne sont limités que par la mémoire disponible pour les stocker.

Les nombreux flottants, en revanche, ont une capacité limitée et ne peuvent représenter qu'une partie des nombres décimaux. Ainsi, des nombres décimaux trop grands ou trop petits ne sont pas représentables. Des nombres comme π, $\sqrt{2}$, qui ne sont pas des nombres décimaux, ne peuvent être représentés que de manière approximative en Python.

Rappel : En mathématiques, un entier relatif est un nombre qui se présente comme un entier naturel auquel on a adjoint un signe positif ou négatif indiquant sa position par rapport à 0 sur un axe orienté.

La division

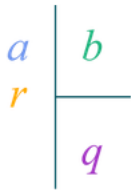
Si on effectue la division de deux entiers avec l'opération /, on obtient un nombre flottant. Ainsi, 7 / 2 donne le nombre 3.5.

Si on veut effectuer en revanche une division entière, alors il faut utiliser l'opération //. Ainsi, 7 // 2 donne le nombre 3, qui est cette fois un entier, à savoir le quotient de 7 par 2 dans la division euclidienne. On peut également obtenir le reste d'une telle division euclidienne avec l'opération %. Ainsi, 7 % 2 donne l'entier 1.

Division euclidienne :

Pour a et b deux nombres entiers (avec b différent de 0), effectuer la division euclidienne de a par b revient à trouver deux nombres entiers q et r qui vérifient l'égalité $a = b \times q + r$ et que $r < b$.

a est le **dividende**.
b est le **diviseur**.
q est le **quotient**.
r est le **reste**.



Variables

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

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

La notation **a =** permet de donner un nom à la valeur à mémoriser. L'interprète calcule le résultat de 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.

```
>>> a
2
```

La variable peut être réutilisée dans la suite des calculs.

```
>>> a * (1 + a)
6
```

Le symbole **=** utilisé pour introduire la variable *a* désigne une opération d'*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.

```
>>> a = 1
>>> a = 2
>>> a = 3
>>> a * ( 1 + a )
12
```

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

```
>>> a = a + 1
>>> a
4
```

- Un nom de variable peut être formé de plusieurs caractères (lettres, chiffres et souligné).
- Il est recommandé de ne pas utiliser de caractères accentués et l'usage veut que l'on se limite aux caractères minuscules.
- Un nom de variable ne doit pas commencer par un chiffre et certains noms sont interdits (car ils sont des mots réservés du langage).

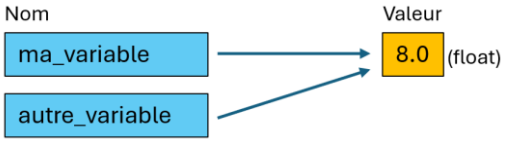
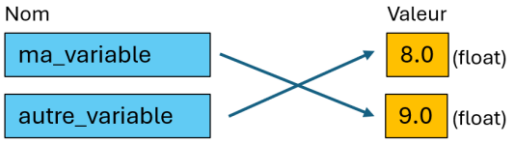
```
>>> cube = a * a * a
>>> ma_variable = 42
```

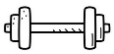
- Il n'est pas possible d'utiliser une variable qui n'a pas encore été définie.
- Seul un nom de variable peut se trouver à la gauche d'une affectation.

Une variable peut être imaginée comme une petite boîte portant un nom et contenant une valeur. Ainsi on peut se représenter une variable déclarée avec x = 1 par une boîte appelée x contenant la valeur 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.

Par exemple, en Python, voici ce qui se passe lors du déroulement de ce script de trois lignes.

Instructions	Représentation de la mémoire à l'issue de l'instruction
ma_variable = 8.0	
autre_variable = ma_variable	
ma_variable = 9.0	

 **Activité :** Exercices 1, 2 , 3 et 4

Etat

À chaque étape de l'interaction, chacune des variables introduites contient une valeur. L'ensemble des associations entre des noms de variables et des valeurs constituent **l'état** de l'interprète Python. Cet état évolue en fonction des instructions exécutées et notamment en raison des affectations de nouvelles valeurs. On dit de ces instructions qui modifient l'état qu'elles ont **un effet de bord**.

On peut représenter l'état de l'interprète par un ensemble d'associations entre des noms de variables et des valeurs.

Par exemple, l'ensemble { a ①, b ② , c ③ } représente l'état dans lequel les variables a,b et c valent respectivement 1, 2 et 3 et où aucune autre variable n'est définie.

3. Mode programme

La mode programme de python consiste à écrire une suite d'instructions dans un fichier et à les faire exécuter par l'interprète Python. Cette suite d'instructions s'appelle un *programme*, ou encore un *code source*. On évite ainsi de ressaisir à chaque fois les instructions dans le mode interactif.

Affichage

En mode programme, les résultats des expressions calculées ne sont plus affichés à l'écran. Il faut utiliser pour ceci une instruction explicite d'affichage. En Python elle s'appelle **print**. Par exemple dans un fichier portant le nom test.py, on peut écrire l'instruction suivante :

```
print(3)
```

On peut alors faire exécuter ce programme par l'interprète Python, ce qui affiche 3 à l'écran. L'instruction print accepte une expression arbitraire. Elle commence par calculer le résultat de cette expression puis l'affiche à l'écran. Ainsi l'instruction :

```
print(1+3)
```

calcule la valeur de l'expression 1+3 puis affiche 4 à l'écran.

Affichage de textes

L'instruction print n'est pas limitée à l'affichage de nombres. On peut lui donner un message à afficher, écrit entre guillemets. Par exemple l'instruction :

```
print("Salut tout le monde ! ")
```

Affiche le message Salut tout le monde à l'écran. Le texte écrit entre guillemets est appelé une ***chaîne de caractères***. Les caractères accentués sont autorisés, tout comme les caractères provenant d'autres langues. On note que les guillemets englobants ne sont pas affichés.

Il est important de comprendre que le contenu d'une chaîne est arbitraire et n'est pas interprété par Python. Pour s'en convaincre, on peut observer la différence entre l'expression arithmétique 1+3 et la chaîne de caractères "1+3" .

En exécutant l'instruction suivante :

```
print("1+3")
```

qui affiche simplement 1+3 à l'écran.

Des guillemets ouverts doivent impérativement être fermés explicitement. On obtiendra sinon une erreur.

Séquence d'instruction

Un programme est généralement constitué de plusieurs instructions. Chaque instruction est écrite sur une ligne. L'interprète Python les exécute l'une après l'autre, dans l'ordre du fichier. Ainsi, l'exécution du programme

```
a = 34
b= 21 + a
print(a)
print(b)
```

affiche deux entiers à l'écran, sur deux lignes successives.

```
34
55
```

Si on souhaite afficher les deux entiers sur une même ligne, on peut remplacer les deux instructions print par une seule, en séparant les deux éléments à afficher par une virgule.

```
print(a,b)
```

Les deux éléments sont affichés sur la même ligne et séparés par un caractère espace.

```
34 55
```

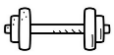
Plus généralement, on peut utiliser print avec un nombre arbitraire d'éléments, qui peuvent être des nombres ou des chaînes de caractères.

```
a = 34
b = 55

print("La somme de", a, " et de ", b, " vaut",  a+b)
```

L'exécution de ce programme affiche le message suivant :

```
La somme de 34 et de 55 vaut 89
```

 **Activité :** Exercice 5

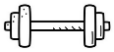
Interagir avec l'utilisateur

Pour permettre l'interaction du programme avec l'utilisateur, par exemple la saisie de la valeur d'une variable, il faut procéder en deux temps : d'abord utiliser l'instruction input pour récupérer des caractères tapés au clavier par l'utilisateur, puis utiliser l'instruction int pour convertir cette chaîne de caractères en un nombre entier.

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

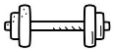
L'instruction input interrompt l'exécution du programme et attend que l'utilisateur saisisse des caractères au clavier. La saisie se termine lorsque lorsqu'il appuie sur la touche entrée. Ici, la suite des caractères saisis par l'utilisateur est stockée dans une variable s. Dans un second temps, la variable a reçoit le nombre représenté par cette suite de caractères. Ainsi, si on exécute ce programme et qu'on saisit successivement les caractères 2, 7 et la touche entrée, le programme affiche la ligne suivante :

```
le nombre suivant est 28
```

 **Activité :** Exercices 6, 7 , 8 , 9

Un programme complet

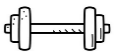
Nous avons maintenant tous les ingrédients pour écrire un programme complet. Il s'agit d'un programme qui demande son année de naissance à l'utilisateur, puis calcule et affiche son âge en 2048. La première ligne est un *commentaire*. C'est une séquence de caractères qui est ignorée par l'interprète et dont le but est uniquement de documenter le programme.

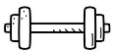
 **Activité :** Compléter le programme suivant.

Programme 1 – calcul de l'âge

```
# calcul de l'âge

saisie = ...
annee = ...
age= ...   # on calcul l'âge par soustraction
print("vous aurez ", age, " ans en 2048. ")
```

 **Activité :** Exercices 10, 11 et 12.

 **Préparer une fiche de synthèse de la leçon ainsi qu'une carte mentale.**
