Objectifs

En suivant les étapes du TP, vous allez voir ou revoir certains concepts en Python et de manière générale en programmation.

Toutes les étapes ne sont pas obligatoires ; vous pouvez vous concentrer sur les éléments que vous ne connaissez pas ou ne vous rappelez pas. Il est néanmoins conseillé de suivre le déroulé afin de garder une certaine cohérence.

Outils

La version de Python utilisée ici est la 3.8. Néanmoins, toute version de Python 3+ devrait être compatible sans problème.

L'éditeur utilisé est IDLE. Néanmoins, n'importe quel éditeur supportant l'édition de code en Python et capable d'envoyer du code à l'interpréteur Python peut être utilisé.

La documentation est obligatoire ; il est impensable de se rappeler de tout, tout le temps, et les autres ressources externes (sites web divers, tutoriaux, vidéos, IA…) ne sauraient jamais être aussi précis et exhaustifs.

Lien vers la documentation : https://docs.python.org/3/

Démarrage d'IDLE et console

Commencez par démarrer IDLE, via la recherche Windows classique ou via le dossier d'installation de Python. L'éditeur est en réalité une console Python, qui vous permet de saisir du code à la volée.

Utilisez Fichier, puis Nouveau fichier afin de créer un script python. Cela vous permettra de conserver votre code et l'exécuter.

Par la suite, nous utiliserons l'expression « dans la console » lorsqu'il s'agira de saisir des éléments dans la console, et « dans le code », « dans le fichier » ou « dans le script » lorsqu'il s'agira de saisir des éléments dans votre code sauvegardé.

Affichage

Tapez l'instruction suivante dans la console : `print("Hello World")`

La **fonction** « print » est utilisée pour envoyer une chaîne vers la **sortie standard**, ici, la **console**.

L'**argument** (ici, un seul) fourni à la fonction, entre parenthèses, est une **chaîne de caractères**, c'est-à-dire une suite de caractères ; on le voit aux guillemets entourant la dite chaîne.

A savoir : on peut utiliser des simples ou des doubles guillemets (ou **quotes**). Il existe aussi d'autres moyens de créer des chaînes de caractères.

```
print('Hello World')
```

L'ensemble de la ligne est nommée « **instruction** ». Une instruction est constituée d'opérations demandées au système.

Sur cette instruction, nous avons demandé au système d'afficher en console un texte.

En python, chaque instruction se termine par un saut de ligne.

Variables

Conserver une information au fil de l'exécution d'un programme est quelque chose de nécessaire. Imaginons, par exemple, que nous souhaitons effectuer un calcul pénible, de la forme $b^2 - 4ac$ $7^2 - 4 * 2 * 6$.

On pourrait faire cela ainsi : `print(7 * 7 - 4 * 2 * 6)`

Cela présente de nombreux soucis (alors qu'on parle que d'une simple opération !) :

- Si on change le 7, on doit le changer deux fois
- En réalité, tout changement va être pénible
- Si on veut réutiliser ce résultat, on doit refaire le calcul

Conserver une donnée en mémoire, durant l'exécution du programme, afin de la réutiliser, c'est tout la notion de **variable** ; on crée une « boîte » nommée, susceptible de contenir un **type de données**, qu'on manipule librement.

Ici, par exemple, nous pouvons écrire notre programme plus proprement ainsi :

```
a = 2
b = 7
c = 6
d = (b*b) - (4 * a * c)
print(d)
```

On voit que la variable d, contenant le résultat peut être réutilisée comme on le souhaite.

Une variable possède :

- Un **nom**
  - que vous décidez
  - qui doit être explicite
  - idéalement en anglais
  - idéalement en snake_case
- Un **type**
  - décidé à la création
  - qui peut se changer « à la volée »

```
x = "a"
x = 5
```

- Une **portée** (c'est le code dans lequel la variable est disponible)
- Eventuellement une **valeur**

Opérateurs

Un **opérateur** sert à faire des opérations entre une ou des opérandes. L'opérateur « + » par exemple, permet d'additionner deux nombres. Mais il peut faire plus.

Dans l'exemple ci-contre, on additionne non pas deux nombres, mais deux chaînes de caractères. Ainsi, le résultat, est une chaîne de caractères « collant » les deux opérandes. C'est une opération de **concaténation**.

```
>>> "a" + "b"
'ab'
```

L'opérateur de division, en programmation (en surtout en python) est le / (slash, ne pas confondre avec l'antislash). Celui de multiplication est le *. Les opérateurs ÷ et × ne sont généralement pas utilisables en programmation.

Les parenthèses vous permettent, comme dans les calculs classiques (et comme vous l'avez connu sur calculatrice), de préciser des priorités de calcul.

L'opérateur d'**assignation**, qu'on a déjà utilisé plusieurs fois, est l'égal (=). Il permet d'assigner une valeur à une variable, de « modifier sa valeur ».

Attention : lorsqu'ils sont doublés, ce sont d'autres opérateurs.

Ainsi, l'opérateur « ** » permet d'élever l'opérande gauche à la puissance de l'opérande droite.

```
>>> 2**3
8
```

Effectuer l'opération de calcul avec a, b, c ci-contre, en utilisant l'opérateur de puissance.

```
a = 2
b = 7
c = 6
d = (b*b) - (4 * a * c)
print(d)
```

Ensuite, calculer les deux valeurs suivantes dans votre programme et affichez-les :

$x_1 = \frac{(-b-d)}{2a}$ $x_2 = \frac{(-b+d)}{2a}$

En réalité, ces formules sont « fausses ». Au lieu de d, nous devrions utiliser la racine carrée de d.

Via des recherches, trouvez comment calculer, en python, une racine carrée, et utilisez cela dans votre code. Attention : il y a deux réponses à cela !

Commentaires

L'opérateur en Python pour noter un **commentaire** sur une ligne est le croisillon (#).

Il est possible de commenter le code, sur n'importe quel langage. Commenter un code permet d'ajouter des précisions. Un bon commentaire explique non pas le code, mais pourquoi il est nécessaire, dans le cas où cela ne serait pas évident. A gauche un exemple de commentaire inutile, polluant. A droite, un exemple de commentaire utile.

```
# je rajoute un
a = a + 1
```

```
# if the request already contains the field, we don't set any default
if field in mapping:
    return
```

Typage

Il est primordial de considérer le **typage** des variables. Même si celui-ci est **dynamique** en python, il est dit **fort** : on ne peut pas faire n'importe quoi entre les types, comme additionner un entier et une chaîne de caractères.

La fonction « type » permet à tout moment de connaître le type d'une variable.

```
>>> type(a)
<class 'int'>
```

Il est important de garder à l'esprit le type de chaque variable, pas seulement parce qu'il peut y avoir des erreurs, mais également dans un souci de mémoire. Chaque type prend une certaine place en mémoire, et il est donc important d'utiliser des types adaptés. Ci-contre, par exemple, est affiché la mémoire utilisée pour le stockage d'une même donnée (le chiffre 5) en tant que chaîne de caractères et en tant qu'entier.

```
>>> sys.getsizeof('5')
50
>>> sys.getsizeof(5)
28
```

Parmi les types existants en Python, quelques-uns sont notables :

Type	Nom	Description
bool	Booléen	Vrai (True, 1) ou faux (False, 0).
int	Entier	Des nombres entiers.
float	Décimal	Des décimaux à précision flottante
str	Chaîne	Du texte. Voilà.
list	Liste	Une liste d'éléments

Il est possible de **convertir** une variable d'un type à l'autre, via des fonctions du même nom. La fonction « str » par exemple renvoie l'équivalent, en chaîne de caractères, de l'argument qu'on lui donne.

Vous vous souvenez du calcul avec a, b et c ?

En utilisant la fonction input(), modifiez votre programme pour demander trois nombres (l'un après l'autre) l'utilisateur : a, b et c. Effectuez le calcul de d, comme auparavant, et affichez le résultat. Un exemple d'utilisation de la fonction input() est indiqué ci-contre.

```
>>> a = input("A = ?")
A = ?5
>>> a
'5'
```

Booléens et comparaison

Toute comparaison est une opération dont le résultat est un booléen. En effet, si je demande si A est supérieur à B, c'est soit vrai, soit faux ; il n'y a que deux résultats possibles, c'est un booléen. Testez le code suivant dans la console : `print(10 > 15)`. Qu'obtenez-vous ? Utilisez type() pour visualiser le type du résultat.

Les opérateurs de comparaisons sont assez intuitifs ; > pour « supérieur », >= pour « supérieur ou égal ». La différence se fait via l'opérateur « != ». L'égalité utilise l'opérateur « == ». Attention à ne pas oublier un égal. Des choses déplaisantes peuvent se produire, sinon.

Structures

Les structures sont des blocs de code. En python, une structure est définie par… des espaces blancs. C'est la grande spécificité de ce langage (la majorité des langages utilisent des accolades pour définir des structures.

Note

Même s'il est possible et historiquement usité d'utiliser des tabulations, depuis quelques années, la bonne pratique en python est d'utiliser exactement 4 espaces blancs par « niveau » de bloc de structure.

Structures de contrôles

Une structure de **contrôle** est une structure dans laquelle le programme « entre » (donc exécute le code dans la structure) si la condition d'entrée est respectée.

Un exemple est le **if**, comme dans l'exemple ci-contre. Observez les espaces à gauche.

```
if d > 0:
    print("Positif")
elif d < 0:
    print("Négatif")
else:
    print("Nul ou non-valide")
```

Si la condition du if est respectée, alors tout le code dans le bloc sera exécuté. Il est possible de rajouter autant de elif que souhaité pour ajouter différents cas qui ne seront exécutés que « sinon si » la condition indiquée est vérifiée. Enfin, il est possible d'ajouter un else qui ne sera exécuté que si aucune autre condition précédente n'a été vérifiée.

Vous vous souvenez du programme avec les nombres a, b et c ?

Désormais, vous devrez vérifier la valeur de d. Si d est supérieur à zéro, alors vous devrez calculer les deux solutions :

$x_1 = \frac{(-b-\sqrt{d})}{2a}$ $x_2 = \frac{(-b+\sqrt{d})}{2a}$

Si d est égal à zéro, vous devrez calculer la solution unique $x_0 = \frac{(-b-d)}{2a}$

Enfin, si d est inférieur à zéro, vous devrez indiquer « aucune solution ».

Boucles

Les boucles ne sont que des structures de contrôles qui s'itèrent (qui se répétent). La boucle **while**, par exemple, se répète tant que la condition indiquée est vérifiée. L'exemple ci-contre est donc une boucle infinie, étant donné que « True » est toujours vrai (c'est le concept même du booléen « vrai »/True).

```
while True:
    print("Au secours")
```

La boucle **for** permet de répéter un nombre de fois fini le bloc d'instructions qu'il contient. Ici, par exemple, cette boucle for se répétera 20 fois, avec la variable point allant de 0 (au premier passage) à 19 (au dernier passage de la boucle).

```
for point in range(0, 20):
    print(point)
```

Toujours sur le même programme, mettez en commentaire ou supprimez la portion demandant a, b et c à l'utilisateur, et créez des boucles afin d'afficher TOUTES les solutions pour a, b, et c compris entre 1 et 10.

Utilisation de bibliothèques

En Python, il est possible d'utiliser des librairies fournies avec le langage ; pour des raisons de performances, il est d'abord nécessaire de charger, d'importer ces librairies au préalable, grâce au mot-clef « import ».

Un exemple, ci-contre, montre qu'il a été nécessaire d'importer la librairie « sys » afin d'utiliser la fonction « sys.getsizeof » (ou la méthode getsizeof de la librairie sys).

```
>>> import sys
>>> sys.getsizeof('a')
```

Il existe un grand nombre de librairies, et il est même possible d'ajouter des librairies qui ne sont pas présentes dans le langage de base. Encore une fois, la documentation regorge d'informations :
https://docs.python.org/3.11/library/index.html

Toujours sur votre programme précédent, transformez le programme pour qu'il choisisse aléatoirement trois nombres a, b et c, et qu'il effectue la résolution.

Créez un nouveau programme.

Demandez à l'utilisateur de saisir deux nombres ; puis affichez leur Plus Grand Commun Diviseur et leur Plus Petit Commun Multiplicateur. Vous allez devoir chercher dans la documentation.