Langages de script (Python)
CMTP n° 1 : Premiers pas en Python

I) Premier programme

Ce premier exercice a pour but de découvrir les différents moyens d'utiliser Python : en mode interactif, en interprétant un fichier `.py`, et en créant un script exécutable.

Exercice 1 :

1. Lancez la version 3 de l'interpréteur Python (mode interactif) grâce à la commande `python3` depuis un terminal. Écrivez et exécutez une commande pour afficher la ligne de texte `Hello World!`. Quittez l'interpréteur en appelant la fonction `exit()`. (*Indications : la fonction qui sert à afficher une chaîne de caractères est* `print`*. On délimite une chaîne de caractères par des guillemets simples ou doubles. Vous pouvez taper* `help(print)` *dans l'interpréteur pour avoir un survol de la spécification de* `print`*.*)
2. Créez un fichier `hello.py` contenant comme unique ligne de texte la commande précédente. Sauvegardez ce fichier, et tapez dans un terminal la commande `python3 hello.py`. Observez le résultat.
3. Rendez le fichier `hello.py` exécutable et tentez de l'exécuter. Que se passe-t-il ? Rajoutez la ligne `#!/usr/bin/env python3` au début du fichier, puis réessayez.

II) Types numériques

Vous pouvez avoir une liste des opérations natives sur les types numériques, `int`, `float` et `complex` ici : https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex.

Exercice 2 : découverte de types différents

Placez-vous dans un interpréteur Python.

1. Faites les calculs `3/4`, `4/2`. Que constatez vous ? Essayez encore `3//4`, `4//2`. Quelle est la différence ? Vous pouvez vérifier votre explication en utilisant la fonction `type`, par exemple en tapant `type(3/4)` et `type(3//4)`.
2. Et si maintenant vous rentrez `type(3j+1)`, comprenez vous la réponse de l'interprète ?
3. Testez les instructions `1j**2`, `(1+2j).imag` et `(1+2j).real`. Que représente `j` ?

Exercice 3 : priorités

1. Calculez le reste de la division de la somme de 19875 et 77569 par 7. Si vous trouvez un nombre supérieur à 6, c'est évidemment qu'il y a une erreur... Que pouvez-vous en déduire sur les opérateurs `+` et `%` ? Qu'en est-il des autres opérateurs arithmétiques (faites des tests) ?
2. Faites le calcul $3^2 + 56/9 \times |-1/4|$, soit en une seule fois, soit en plusieurs fois en utilisant des variables.

La liste des priorités des opérateurs en Python est disponible ici : https://docs.python.org/3/reference/expressions.html#operator-precedence.

Exercice 4 : conversions

1. Tapez `hex(30)`. Quel format est utilisé pour représenter les hexadécimaux ? Les octaux ? Les chaînes binaires ?
2. La fonction `int(str,int)` permet de faire la conversion de base. Le deuxième argument est optionnel, dans ce cas alors la valeur par défaut étant 10. Quel est le résultat des expressions suivantes ? `int('101',2)`, `int('101',8)`, `int('101',10)`, `int('101', 16)`, `int('101')`.
3. Écrivez une expression qui convertit la chaîne binaire '111001' à une chaîne représentant la même valeur en hexadécimal.
4. Consultez l'aide de la fonction `input` (vous pouvez taper `help(input)` sur le mode interactif, ou chercher une bonne explication sur le Web). Écrivez une suite d'instructions permettant de lire un entier puis d'afficher le double de sa valeur.

III) Chaînes de caractères

Vous trouverez une introduction aux chaînes de caractères en Python ici https://docs.python.org/3/tutorial/introduction.html#strings.
En particulier :

- `s+t` concaténation des chaînes `s` et `t`,
- `s*n` concaténation `n` fois de la chaîne `s`,
- `s[n]` accès au caractère (i.e. chaîne de longueur 1) d'indice `n` (à partir de 0) de la chaîne `s`,
- `s[start:stop:step]` slices (découpages) pour extraire de `s` une chaîne de certaines de ses éléments. À reprendre dans les prochaines séances.

Exercice 5 : opérations de base

1. Créez une variable `h` contenant la chaîne de caractères `"Hello"` et une variable `w` contenant `"World"`. En utilisant la concaténation de chaînes de caractères et ces deux variables (entre autres), créez une nouvelle variable `hw` contenant la chaîne `"Hello␣World!"`.
2. Comparez le résultat des instructions `hw` et `print(hw)`. Quelle est la différence ?
3. Quel est le résultat de l'expression `h+2*(h+2*w)` ? Pourquoi ?

Exercice 6 : caractères spéciaux

1. Créez une variable contenant la chaîne de caractères suivante (en respectant les retours à la ligne) :

   Dans le vieil étang,
   Une grenouille saute,
   Bruit dans l'eau.

   Utilisez d'abord des guillemets triples (`'''` ou `"""`), puis des guillemets simples.
2. Créez une variable contenant la chaîne suivante :

   Les chaînes sont entourés par " ou '
   le caractère spécial \n représentant un retour à la ligne

Exercice 7 : fonctions auxiliaires

1. Testez la fonction `len(str)`. À quoi sert-elle ?
2. Testez les instructions : `"e" in "hello"`, `"E" in "hello"`, `"ell" in "hello"`, `"hl" in "hello"`. À quoi sert l'opérateur `in` dans ce contexte ?

Exercice 8 : slices

Affectez à la variable `s` la chaîne `'0123456789'`.

1. Testez ensuite les instructions suivantes en expliquant leur comportement :
   (a) `s[4]`
   (b) `s[-4]`
   (c) `s[:4]`
   (d) `s[-4:]`
   (e) `s[:]`
   (f) `s[:4] + s[4:]`
2. En n'utilisant que la syntaxe des "slices", donnez des expressions qui affichent :
   (a) un caractère sur deux de `s`
   (b) les caractères de `s` dans l'ordre inverse.
   (c) un caractère sur deux en partant de la fin, dans l'ordre inverse.

IV) Booléens, instructions conditionnelles

Une *expression booléenne* (type `bool`) peut être évaluée en `True` ou `False`, ce qui permet l'alternative entre les deux branches d'une instruction conditionnelle. Une expression booléenne peut être générée, entre autre, par des comparaisons, comme `==`, `!=`, `<`, `<=`, `>`, `>=` et peut être composée par les opérateurs logiques `and`, `or` et `not`.

Exercice 9 :

Simplifier une expression booléenne consiste à la remplacer par une autre expression booléenne équivalente qui est plus courte. Simplifier les expression suivantes :

- `x == y and x == z and y == z`
- `x == 17 or (x != 17 and x == 42)`
- `x > 5 or (x <= 5 and y > 5)`
- `not (x > 3 and x == 5) or not (x < 5 or x > 5)`

Les syntaxe basique pour une instruction conditionnelle en Python est la suivante :

```
if condition:
    instructionA_1
    ...
    instructionA_n
else:
    instructionB_1
    ...
    instructionB_m
```

où `condition` est une expression booléenne. Notez bien que *l'indentation est obligatoire !* et, de préférence, réalisée en utilisant *4 espaces* pour chaque niveau d'indentation. Python en effet utilise l'indentation pour délimiter un bloc d'instructions et son dépendance vers d'autres constructions, comme les instructions conditionnelles et les boucles.
La branche négative d'une conditionnelle (introduite par `else`) est *optionnelle*. En cas d'une conditionnelle imbriquée sur la branche `else`, on peut utiliser la concaténation `elif` suivie d'une condition :

```
if conditionA:
    instructionA_1
    ...
    instructionA_n
elif conditionB:
    instructionB_1
```

```
    ...
    instructionB_m
else:
    instructionC_1
    ...
    instructionC_k
```

On peut avoir un nombre arbitraire de branches `elif` qui suit une branche `if`.

Exercice 10 :
Écrire un programme qui demande à l'utilisateur un entier et affiche `"positif"` si cet entier est strictement supérieure à zéro, affiche `"negatif"` si strictement inférieure à zéro et affiche `"zero"` si l'entier vaut zéro.

V) Boucles et itérables

La syntaxe de base des boucles `for` de Python est la suivante :

```
for x in iterable:
    instruction_1
    ...
    instruction_n
```

où `iterable` est une structure de données sur laquelle on peut "itérer", c.-à-d. une collection dont on peut visiter tous les éléments exactement une fois dans un ordre spécifique. Intuitivement, on peut toujours demander quel est le prochain élément à visiter dans un itérable (cela sera la valeur passée à `x` par le `for` ci-dessus), tant qu'il y a des elements encore à visiter. Plus en détail, nous n'avons pas un accès direct à cette méthode du "prochain élément" (sauf si l'itérable est aussi un itérateur, comme on verra dans la suite du cours) et on doit utiliser le mot clé `for` pour cela.
Les chaînes de caractères sont un exemple d'itérable, qui donne un caractère (c.-à-d. une chaîne de longueur 1) à chaque itération. Nous verrons d'autres exemples d'itérables (listes, n-uples, les intervalles, les fichiers, les dictionnaires, les générateurs, etc) dans la suite du cours.

Exercice 11 :
Testez le code suivant :

```
s = "Hello, World!"
for c in s:
    print(c)
```

Modifiez ce code pour qu'il affiche que les voyelles de la chaîne `s`.

Un autre type d'itérable très utile pour les boucles sont les *intervalles* d'entiers, générés par le constructeur `range(start, stop[, step])`.

Exercice 12 :
Testez les codes suivants :

```
for x in range(0, 10):
   print(x)

for x in range(0, 10, 3):
   print(x)

for x in range(9, -1, -1):
   print(x)
```

Comprenez vous la spécification de `range` ? Vous pouvez confirmer vos hypothèses en lisant https://docs.python.org/3/tutorial/controlflow.html#the-range-function.

Exercice 13 :
Écrivez une programme exécutable qui demande à l'utilisateur de rentrer un entier `n` et affiche la somme des `n` premiers nombres paires, en utilisant une boucle `for` et `range`.

Bien sur les boucles peuvent être imbriqués, en prenant soin de l'indentation afin de préciser l'appartenance d'une instruction au corp d'une boucle.

Exercice 14 :
Écrivez une programme exécutable qui demande à l'utilisateur de rentrer un entier `n` et ensuite affiche un triangle d'étoiles sur `n` lignes. Par exemple, si `n` vaut 5, le programme affiche :

```
*
**
***
****
*****
```

La syntaxe basique pour une boucle `while` est la suivante :

```
while condition:
    instruction_1
    ...
    instruction_n
```

où `condition` est une expression booléenne.

Exercice 15 : Syracuse.
La suite de Syracuse de premier terme p (entier strictement positif) est définie par $a_0 = p$ et

$$a_{n+1} = \begin{cases} \frac{a_n}{2} & \text{si } a_n \text{ est pair} \\ 3 \cdot a_n + 1 & \text{si } a_n \text{ est impair} \end{cases}$$

Une conjecture (jamais prouvée à ce jour !) est que toute suite de Syracuse contient un terme $a_m = 1$. Écrire un programme exécutable qui demande à l'utilisateur un entier p et affiche tous les termes de la suite de Syracuse de premier terme p jusqu'au premier terme qui vaut 1.

VI) Fonctions

La syntaxe de base pour définir une fonction en Python est la suivante :

```
def nom_fonction(arg_1,..., arg_m):
    "une chaîne (optionelle) décrivant la fonction"
    instruction_1
    ...
    instruction_n
    return resultat
```

C'est possible de définir des paramètres optionnels ou par mot clés, voir https://docs.python.org/3/tutorial/controlflow.html#more-on-defining-functions pour plus de détails.

Exercice 16 : Fonctions sur les chaînes
Écrire les fonctions

1. `count_blanks(s)` qui renvoie le nombre d'espaces dans la chaîne de caractères `s` passée en argument ;
2. `is_palindrome(s)` qui retourne `True` si la chaîne de caractères `s` passée en argument est un palindrome, et `False` sinon (*Indication : penser à utiliser les slices*) ;
3. `count(s, o)` qui retourne le nombre d'occurrences *non-superposées* des sous-chaînes de `s` égal à `o` (*Indication : penser à utiliser les slices*).

Vérifiez bien vos fonctions avec des tests bien choisis. Par exemple, `count("aljjsdfajajjjj", "jj")` doit renvoyer `3` et `count("jjj", "jj")` `1`.

VII) Bibliothèque standard et modules

Vous pouvez utiliser les fonctions définies dans des modules de la bibliothèque standard de Python : https://docs.python.org/3/library/. Nous étudierons certains de ces modules dans la suite de cours. Le module `math`, par exemple, contient toutes les fonctions mathématiques définies par la norme C, voir https://docs.python.org/3/library/math.html.
En général, pour utiliser les fonctions définies dans un module il faut d'abord importer le module par l'instruction `import`, par exemple `import math`. Il est courant de ranger tous les import au début d'un fichier. Ensuite, vous pouvez appeler les fonctions via le nom du module, par exemple `math.sqrt(4.0)`.
Il existe aussi d'autres variantes de l'instruction `import` permettant d'éviter la notation pointée :

- `from math import sqrt` charge du module `math` que la fonction `sqrt`, cette dernière pouvant être appelée tout simplement avec son nom, par exemple `sqrt(4.0)`,
- `from math import *` charge toutes les fonctions du module `math`, qui seront appelées tout simplement avec leur nom : à éviter car il peut générer des conflits de noms.

Exercice 17 : Speedy Gonzales

En utilisant le module `time` (doc ici : `http://docs.python.org/3/library/time.html`) comparer le temps d'exécution de votre implémentation de la fonction `count` de l'exercice 16 avec celui de la fonction homonyme de la classe `str` (voir la doc ici :
https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str).
Remarquez qu'il n'y a pas besoin d'importer le module `string` car la classe `str` est native en Python3. Remarquez aussi que `count` est une méthode d'instance de la classe `str` : l'appel `count(s, o)` de l'exercice 16 corresponde à l'appel `s.count(o)` de la classe `str`. Nous reviendrons sur la différence entre méthodes de classe vs méthodes d'instances dans la suite du cours. Qui est plus rapide sur des entrées comme `s = "Speedy␣Gonzales"* 10000` et `o = "ee"` ? Savez vous faire mieux ?