TP 0| Premiers pas avec Python

1. Prise en main et présentation de Spyder

Aller dans votre répertoire personnel, et créer un répertoire Informatique. Dans celui-ci, créer un autre répertoire TP0.

Ouvrez Spyder qui sera l'environnement de développement dont nous allons nous servir pendant l'année, pour travailler sur Python. Normalement, à l'ouverture de Spyder, vous avez quelque chose qui ressemble à la figure 1.

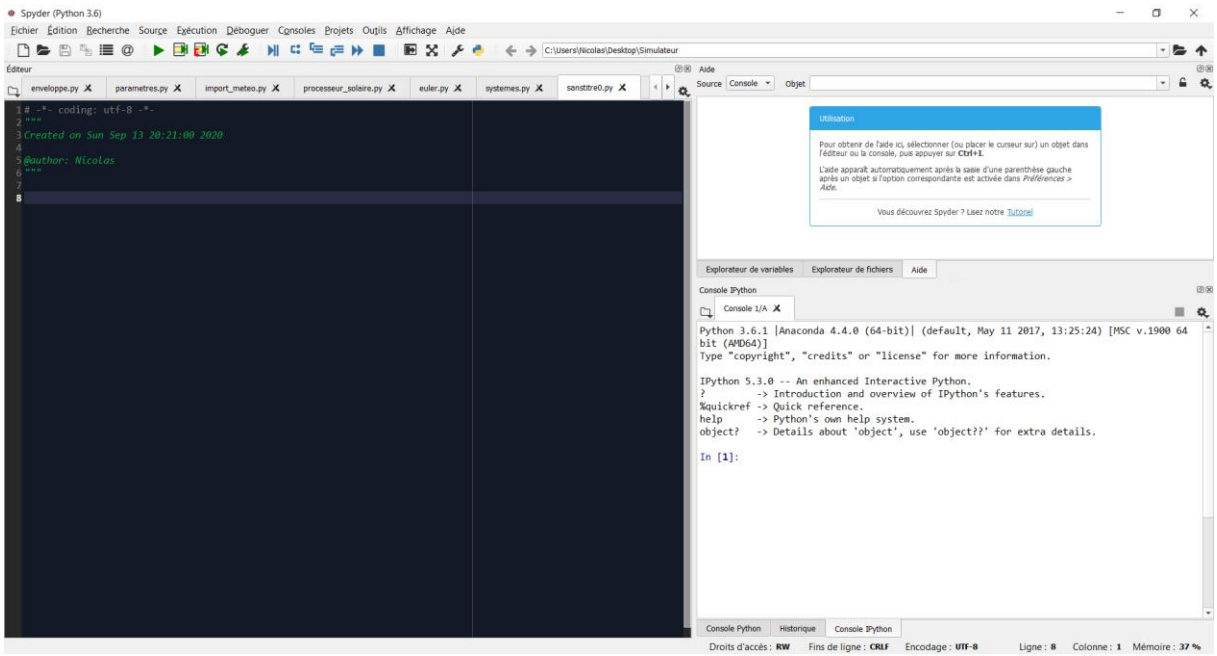

Figure 1. L'environnement de développement Spyder

Il y a trois parties distinctes :

- la partie en bas à droite est la console Python. On peut directement écrire du code Python dedans (au niveau du In [...]), il est alors interprété immédiatement ;
- la partie à gauche est l'éditeur : cette partie sert à rédiger des scripts Python plus complexes (ouvrez une nouvelle page avec Nouveau fichier ou CTRL+N). Une fois le script écrit on « l'envoie vers la console » (Exécuter le fichier) ou F5.
- la partie en haut à droite comprend l'aide, l'explorateur de fichiers mais également l'explorateur de variables que vous aurez créées.

1.1 La console : quelques calculs

On se concentre ici sur la console Python. Positionnez-vous au niveau de la première fenêtre, qui est la console Python, plus précisément au niveau du In[numéro de ligne], qui s'appelle « l'invite de commandes ». Dans le texte qui suit, on le remplacera par trois chevrons >>>. La console permet de lancer des commandes, immédiatement interprétés par Python. Commençons par le classique :

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

que vous ferez suivre de la touche [ENTRÉE]. Lancez ensuite quelques calculs simples (suivis de la touche [ENTRÉE]), comme ceux-ci :

```
3+4    -5.5*6    3.0-4*5    13//4    13/4    13%4    5**3    (5+2)*(5-2)
```

En Python 3, // est la division entière (quotient dans une division euclidienne), % le modulo (le reste dans cette division), et ** l'exponentiation (puissance). Ainsi, on peut utiliser la console comme une grosse calculette. On peut également utiliser des variables : tapez les trois lignes suivantes, à chaque fois suivies de la touche [ENTRÉE].

```
>>> a=6+3
>>> b=a//2
>>> print(a+b)
```

Remarquez que = est utilisé pour affecter une valeur à une variable.

1.2 Enregistrement d'un script

Dirigez-vous maintenant vers la partie éditeur (qui doit être une page blanche, appelée <sanstitre0>) ou ouvrez-en une autre avec Fichier > Nouveau fichier. Enregistrez la page sous le nom TP0_1.py, dans le sous-répertoire TP0 que l'on a créé tout à l'heure.

1.3 Exécution d'un script

Écrire maintenant dans le fichier le code suivant (les espaces sont obtenus par une tabulation).

```
a=3
if a%2==0:
	print("a est pair")
else:
	print("a est impair")
```

Remarquer que Spyder vous aide en plaçant pour vous les tabulations (on parle d'indentation automatique). Lorsque vous voulez revenir en début de ligne (par exemple pour taper le else ou le print), utilisez simplement la touche de retour chariot.

Sauvegardez votre code (CTRL+S ou Fichier > Enregistrer à la souris), puis appuyez sur F5 ou faîtes Executer le fichier. Débrouillez-vous maintenant pour faire subir à une variable b initialisée à 4 le même traitement que l'on a fait subir à a (un copier-coller est tout indiqué !). Appuyez sur F5 et vérifiez le résultat dans la console.

1.4 Les commentaires

Il est important, du point de vue de la réutilisabilité du code, de mettre des commentaires : un morceau de code ayant une utilité tout à fait claire au moment où vous l'écrivez, vous paraîtra beaucoup plus obscur dans plusieurs mois. Cela peut aussi servir pour travailler à plusieurs sur un fichier. Les commentaires sont des morceaux de code qui ne sont pas interprétés par Python, et sont destinés à l'humain seulement. Vous pouvez aussi commenter les codes dont vous n'avez plus besoin, pour éviter par exemple d'avoir des affichages à l'écran qui ne sont plus utiles.

Il y a plusieurs manières de mettre un commentaire dans le code. Premièrement, sur une seule ligne, tout ce qui suit le caractère dièse (#) est un commentaire. C'est ce qu'on peut utiliser par exemple pour expliquer ce que fait une variable. Pour commenter plusieurs lignes, on peut mettre # en chaque début de ligne (sous Spyder : sélectionner une région, et faire Edition > Commenter ou CTRL + 1, idem pour décommenter). Une autre solution est d'utiliser des chaînes de caractères pouvant s'écrire sur plusieurs lignes : mettre trois « quotes » (le caractère ") avant et après la partie à commenter. Par exemple :

```
"""
Ceci est un commentaire, vous pouvez le taper et faire F5 pour tester.
"""
a=4 #Ce qui suit le # est aussi un commentaire. On affecte à a la valeur 4.
```

2. Les types simples

En Python, les objets ont des types, qui caractérisent les attributs de l'objet, ainsi que les fonctions que l'on peut appliquer dessus. On appelle type simple l'un des types suivants : entier, flottant, booléen. Rappelez-vous que type(valeur) donne le type de la valeur passée en argument.

2.1 Présentation des types simples

Tapez les lignes suivantes dans votre fichier TP0_1.py, et exécutez-les.

```
a=1
b=a/2
c=a<b
print(type(a))
print(type(b))
print(type(c))
```

On retrouve les types int (pour integer, entier en français), float (pour floating point number, nombre à virgule flottante ou simplement flottant en français), bool (pour boolean, booléen). Si vous êtes choqués par la ligne c=a<b, voici une explication : a<b est une comparaison entre a et b. Son résultat est soit vrai (True) soit faux (False), et le résultat de cette comparaison est stocké dans c. Vous pouvez taper c dans la console pour vérifier sa valeur.

2.2 Opération sur les entiers/flottants

On donne ci-dessous la signification des opérateurs mathématiques élémentaires sur les entiers ou les flottants (nombres à virgule arrondis).

Opérateur	+	-	*	/	//	%	**
Signification	Addition	Soustraction	Multiplication	Division	Division entière	Modulo	Puissance

Remarquez que les entiers sont automatiquement convertis en flottants, si besoin.

La règle de priorité des opérateurs dans les expressions arithmétiques est la suivante : on commence par évaluer les puissances, puis les modulos, ensuite les multiplications et divisions, et enfin les additions et soustractions. Pour les cas d'égalité de priorités, les opérateurs les plus à gauche sont évalués en premier (Sauf l'exponentiation : testez 2**2**3). Les parenthèses sont à votre disposition pour changer l'ordre des priorités.

2.3 Booléens et opérateurs de comparaison

Les booléens ne prennent que deux valeurs : True et False. Ils sont utilisés essentiellement comme conditions : on a par exemple utilisé une structure conditionnelle (if) tout à l'heure, testant l'égalité d'un entier avec 0. Suivant la valeur du booléen, l'une ou l'autre partie du code était exécutée. Voici la liste des opérateurs de comparaison sur les entiers/flottants :

<	>	<=	>=	==	!=
Strictement inférieur	Strictement supérieur	Inférieur ou égal	Supérieur ou égal	Egal	DIfférent

De même que l'on peut faire des opérations entre entiers et flottants, on peut faire des opérations entre booléens. Par exemple, pour exprimer que l'entier n est strictement positif et inférieur ou égal à 100, on écrira : n>0 and n<=100 (On peut remarquer que Python comprend aussi très bien 0<n<=100...). Le tableau suivant présente les différents opérateurs booléens et leurs *tables de vérité*.

a	b	not a	a or b	a and b
False	False	True	False	False
False	True		True	False
True	False	False	True	False
True	True		True	True

L'ordre de priorité d'évaluation pour les opérations booléennes est not, puis and et enfin or. On peut également, comme sur beaucoup d'objet Python, utiliser les opérateurs == et != pour l'égalité et la différence.

3. Importer un module (et voir l'aide)

Prenons deux opérations mathématiques classiques : la racine carrée et l'exponentielle. En Python, elles sont données par sqrt (square root) et exp. Seulement, ces fonctions nécessitent d'être importées de la bibliothèque standard, car elles ne se trouvent pas dans le *noyau Python*. Sans surprise, on les trouve dans un module appelé math. Deux (quatre) solutions pour les importer :

- importer le module math avec la commande import math. On accède ensuite aux fonctions avec **math.sqrt** et **math.exp**.

- une variante de la précédente : on peut utiliser un alias pour éviter de taper le nom du module en entier. Par exemple, on importera le module sous le nom m, avec **import math as m**. On accède ensuite aux fonctions avec **m.sqrt** et **m.exp**. C'est particulièrement utile lorsque les noms des modules sont longs.
- importer spécifiquement les fonctions sqrt et exp du module math, en tapant from math import sqrt, exp. On accède aux fonctions simplement avec sqrt et exp.
- une variante de la précédente : importer toutes les fonctions du module math avec **from math import ***. Cette solution est à éviter pour plusieurs modules : on risque d'avoir plusieurs fonctions avec le même nom. Cela dit, on peut le faire sans problème avec un seul module, et le module math s'y prête bien.

```
import math # puis utilisation de math.sqrt et math.exp

import math as m # puis utilisation de m.sqrt et m.exp
from math import sqrt,exp # puis utilisation de sqrt et exp
from math import * # puis utilisation de sqrt et exp
```

Il y a bien sûr d'autres fonctions usuelles dans le module math. Outre les fonctions trigonométriques et logarithmiques qui ont les noms attendus, on trouve par exemple les définitions des constantes e et pi, qui s'importent de la même façon. On verra quelques autres packages pendant l'année. Comme il y en a beaucoup, il est hors de question de connaître tous ces packages et les fonctions qu'ils contiennent par cœur. Python propose une documentation très détaillée à l'adresse : *http://docs.python.org/3/*.

4. A vous de jouer !

Exercice 1. Type des expressions.

Prévoir le type des expressions suivantes, et le résultat.

1) 2**3.0+4

2) 2>=3 or 2**4*5**2//20==20

3) True or 4>3 and 3>4

4) 5%3*5

5) not False and False

6) int(8.6)+2 Que fait int ?

7) float(2)**3 Que fait float ?

Exercice 2. Expressions booléennes.

1) Écrire des expressions booléennes exprimant le fait que les quantités qui suivent sont positives. Vérifiez avec Python si elles s'évaluent en True ou False. N'oubliez pas d'importer ce dont vous avez besoin du module **math**.

$$\sqrt{1+\sqrt{1+\sqrt{1+1}}}-\frac{1+\sqrt{5}}{2}$$

$$\frac{\sqrt{3}}{2}-\frac{e}{\pi}$$

2) Donner une expression booléenne dépendant des variables a, b, c et d exprimant le fait que a est supérieure ou égale à b, c et d. Tester éventuellement en affectant préalablement des valeurs aux variables.

3) Soit $\Omega = (x, y)$ un point du plan, $r > 0$ et $A = (x_A; y_A)$ un autre point. On rappelle que la distance de A à Ω est donnée par $\sqrt{(x-x_a)^2+(y-y_a)^2}$, ce que vous vérifierez rapidement avec un dessin au papier. Déterminer une expression booléenne dépendant des variables X, Y, x_A, y_A et r permettant de tester si le point A est strictement à l'intérieur du cercle de centre Ω et de rayon r. Tester votre condition avec les valeurs suivantes.

$$\Omega = (0,0),\ \ r = 2,\ \ A = (1,1.5)$$

$$\Omega = (1,3),\ \ r = 4,\ \ A = (5,2)$$

4) Reprendre la question précédente pour écrire une condition **if** comme dans l'exemple situé plus haut, disant si oui ou non A est à l'intérieur du cercle. Remarque : si b est un booléen, il est tout à fait valable d'écrire **if b:**, alors qu'il est redondant d'écrire **if b==True:**. On utilisera la fonction d'affichage **print**, appelée avec une chaîne de caractères comme "A est à l'intérieur du cercle".

Autour des boucles for

Pour $n < m$ deux entiers, on rappelle que **range(n,m)** produit un itérable, fournissant la suite des entiers de n à m - 1. On fait parcourir à une variable i ces entiers-là à l'aide d'une boucle for ayant la forme suivante :

```
for i in range(n,m):
      [instructions]
```

Par exemple :

```
s=0
for i in range (1,10):
      s=s+i #s contiendra la somme des entiers de 1 à 9
```

Les instructions indentées étant répétées pour tout i entre n et m - 1. On peut de plus spécifier un pas p : pour p > 0, **range(n,m,p)** permet d'obtenir les entiers n, n + p, n + 2p... strictement inférieurs à m.

Exercice 3. Quelques boucles for dans des calculs des sommes.

1) Vérifiez à l'aide de Python que $1 + 2 + \cdots + 100 = \frac{100\times101}{2}$. Proposez une formule pour $1 + 2 + \cdots + n$, vérifiez-la pour quelques valeurs de n avec Python. Comment la démontrer?

2) Vérifiez à l'aide de Python que $1 + \frac{1}{2} + (\frac{1}{2})^2 + \cdots + (\frac{1}{2})^{10} = \frac{1-(\frac{1}{2})^{11}}{1-\frac{1}{2}}$. Proposez une formule pour $1 + q + q^2 + \cdots + q^n$, vérifiez-la pour quelques valeurs de q et n .

3) Vérifiez expérimentalement la formule suivante (dure à démontrer) :

$$\frac{\pi^2}{6} = \left[\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{n^2} + \cdots\right] = \sum_{n=1}^{+\infty} \frac{1}{n^2}$$

On coupera la somme après un « grand » nombre de termes, comme 10000 ou 100000). Ces sommes infinies, appelées « séries », seront étudiées au second semestre de mathématiques. Voici quelques autres formules « mystérieuses » que l'on peut vérifier avec Python :

$$\ln(2) = \left[1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + \frac{(-1)^n}{n+1} + \cdots\right]$$

$$\frac{\pi}{4} = \left[1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + \frac{(-1)^n}{2n+1} + \cdots\right]$$

$$\frac{\pi}{8} = \left[\frac{1}{1\times3} + \frac{1}{5\times7} + \frac{1}{9\times11} + \cdots + \frac{1}{(4k+1)(4k+3)} + \cdots\right]$$

Autour des boucles while

On rappelle le fonctionnement d'une boucle **while** : tant qu'une certaine condition est vérifiée, on réalise les instructions de la boucle. Plus précisément, à chaque fois que l'expression booléenne

s'évalue en True, on fait un tour de boucle et on réévalue la condition. Dès qu'elle s'évalue en False, on passe aux instructions situées après la boucle.

```
while expression:
      [instructions]
```

Par exemple :

```
a=123
while a>0:
      print(a) #on affiche à l'écran les quotients dans les
      a=a//2   #divisions successives de 123 par 2
```

Dans cet exemple, la boucle while s'arrête dès que la variable a ne vérifie plus a>0, ce qui arrive au bout d'un moment car cette variable décroit strictement à chaque itération.

Lorsque l'on fait une boucle while, il est important de mettre une condition qui est vérifiée au bout d'un moment (la boucle « termine »). Nous étudierons cela dans le chapitre dédié.

Exercice 4. La conjecture de Syracuse (un des problèmes les plus difficiles des maths!)

Pour $u_0 \in \mathbb{N}^*$, on considère la suite définie par :

$$u_{n+1} = \begin{cases} 3u_n + 1 & si\ u_n\ est\ impair \\ {u_n}/{2} & sinon \end{cases}$$

On conjecture que pour tout $u_0 \in \mathbb{N}^*$, il existe un indice n tel que $u_n = 1$ (à partir de cet entier n, la suite prend périodiquement les valeurs 1; 4 et 2).

1) Écrire une boucle **while** permettant de vérifier cette conjecture pour un u_0 fixé. Testez-là pour différentes valeurs de u_0.

Remarque : on n'a ici besoin que d'une variable pour calculer successivement les termes de la suite, qu'on pourra appeler u.

2) On appelle temps de vol à partir de u_0 le plus petit n tel que u_n vaut 1. Calculez ce temps de vol pour $u_0 = 15$ et $u_0 = 127$. On affichera les valeurs de la suite obtenues avant d'atteindre 1, à l'aide de **print**.

Exercice 5 (divers).

1) On exécute le code suivant, donner les valeurs des variables après exécution.

```
a=5
b=a+2
a=b
c=a==b
a=10
d=a!=b
```

2) On suppose x, y et z affectées, ces variables sont des entiers. Indiquer un code permettant de tester si au moins deux des entiers sont égaux (on affichera un message à l'écran, du style « il y en a deux égaux » ou « ils sont tous distincts »).

3) Soit $n \in \mathbb{N}^*$. On appelle *valuation 2-adique* de n l'unique entier p tel que $n = 2^p m$, avec m un entier impair. Par exemple, la valuation 2-adique de 7 est 0, celle de $96 = 2^5 \times 3$ est 5. On suppose la variable n affectée, écrire un code utilisant une boucle **while** permettant d'obtenir la valuation 2-adique de n, et de l'affecter dans la variable p.