TP 1 : Introduction à Python

I Introduction

Au cours de vos études en CPGE, vous allez apprendre un langage informatique nommé Python. Pour coder en Python, on utilisera un EDI (pour environnement de développement intégré) nommé `Spyder`*. L'objectif étant d'arriver aux concours avec des bases **solides** vous permettant de résoudre des problèmes à l'aide d'algorithmes que vous avez programmé.

II Variables

Une **variable** permet de **stocker** une information dans la **mémoire** de votre machine. Pour cela, il faut deux choses le **nom** de la variable et sa **valeur**. Voici, plusieurs exemples, tapez-les dans l'éditeur de script et exécutez le script.
La commande **print** permet **d'afficher** la valeur de la variable.

```
a = 2 #Crée une variable notée a dont la valeur est 2
nd = 0.5
print("La valeur de a vaut",a)
a = (a+2*nd)/3#Modifie la valeur de a mais pas celle de
    nd
print("La valeur de a vaut",a)
L = [1,2,8]#L est une liste qui contient 3 éléments: 1,
    2 et 8
print("La liste L est", L)
s = "les maths, c'est génial"#s est une chaîne de caract
    ères
print(s)
```

Remarque 1.
- ▷ On peut évidemment effectuer les opérations usuelles sur les variables qui sont des nombres.
- ▷ Pour savoir de quel type sont les variables, utiliser la commande **type**, essayer : les commandes **type(a), type(s), type(c), type(0.2)**.
- ▷ Vous pouvez aussi essayer de voir ce que donne l'addition d'un nombre avec une liste, ou l'addition de deux listes.

```
a = 22
b = 3
c = a-b#Que vaut c?
q = a//b#Quotient de la division de a par b
r = a%b#Reste de la division de a par b, Calculer q et r
#de tête avant de vérifier le calcul par Python.
#Rappel: a=bq+r avec 0 <= r< b.
```

- ▷ Comprenez que si `a` et `b` sont deux variables déjà définies, alors les instructions `a=b` et `b=a` ne sont pas du tout équivalentes. La première modifie la valeur de la variable `a` qui prend la valeur de `b`, la seconde fait l'inverse.
- ▷ Le «=» est en fait ici un **symbole d'affectation** de valeur†.

Exercice 1 ★☆☆

Prédire ce que seront les valeurs des variables `a` et `b` à la fin du script suivant, puis taper le code pour vérifier votre prédiction.

```
a = 10
b = 5
a = a+b
b = a+b
b = a
b = 2*b
```

Exercice 2 ★☆☆

Étant données deux variables `a` et `b`, écrire un script qui échange les valeurs de a et b (*la commande* `a,b=b,a` *est bien évidemment interdite*). Tester si cela fonctionne.

III Fonctions

Comme en mathématiques, une **fonction**, en Python, renvoie un résultat qui dépend des **arguments** donnés en **entrée**.

*. une alternative étant l'IDE appelé Pyzo

†. Certains langages notent `a←b` pour souligner que la variable `a` va changer de valeur et prendre celle de `b`. On lit «a reçoit b» ou encore «on affecte (la valeur de) b à a»

- On **définit** une fonction en expliquant à la machine l'**algorithme** de la fonction.
  Analogie en mathématiques : «Pour tout $x > 0$, $f(x) = x^2 - 1$.»
- On **appelle** une fonction lorsqu'on l'utilise dans un programme.
  Analogie en mathématiques : «Considèrons $a = f(5)$.»
- Contrairement aux fonctions vues au lycée, une fonction Python peut prendre un ou plusieurs arguments, *de même type, ou non*! Il est possible de n'avoir aucun paramètre, dans ce cas on laisse les parenthèses sans rien dedans `def MaFonction():`
- ♥ La définition d'une fonction commence toujours par un `def` et se termine toujours par un `return`! On n'oublie pas les **deux points** après avoir donné le nom de la fonction et ses éventuels arguments.
- Les instructions qui sont dans **le corps** de la fonction doivent subir une indentation.
- Une fonction peut ne rien renvoyer. Dans ce cas, on termine par `return` tout simplement. Cela signifie à la machine que le corps de la fonction est terminé.

```
23 def FonctionMystere(a,b,c):
24     s = a+b+c
25     m = s/3
26     return m
```

Dans cet exemple, la fonction `FonctionMystere` prend plusieurs arguments en entrée : `a, b` et `c`. Cette fonction renvoie la valeur `m`.

Exercice 3 ★☆☆

Taper ces lignes et exécuter. Que se passe-t-il? Essayer maintenant la commande suivante et indiquer ce que calcule la fonction mystère :

```
27 Resultat = FonctionMystere(5,10,14)
28 print(Resultat)
```

⚠ Il ne faut pas confondre **`return`** et **`print`**! La commande `print` *affiche* la valeur d'un calcul mais ne la stocke pas dans une variable, pour cela, en fin de fonction, il faut utiliser `return`. Dans un premier temps, vous pouvez naïvement garder en tête qu'«une fonction commence par un `def` et se termine par un `return`».

IV Conditions

IV.1 Définition

Les conditions permettent d'effectuer certaines instructions uniquement dans certains cas. Par exemple, si a et b sont des nombres, on effectuer une instruction que si $a > b$ ou si $a \geq b$ ou encore si $a = b$. Comment décrire ces cas en Python?

Recopier ce code, et exécutez le plusieurs fois, en faisant changer la valeur de la variable `age`.

```
29 age = 17#La variable age vaut 17
30 if age == 18:#Le == est indispensable pour ne pas
31 # confondre avec l'affectation de variable age = 18
32     print("Tu es à peine majeur")
33 if age >= 18:
34     print("Tu es déjà vieux")
35 if age < 18:
36     print("Tu es un bébé")
37     attente = 18-age
38     print("Attends ",attente," année(s) avant ta majorit
   é")
39 if age != 0 and age < 18:#!= veut dire différent
40     print("Tu as vécu au moins une année mais tu es
   mineur")
```

Définition 1

Une **condition** est une expression dont le résultat est soit « vrai » soit « faux ». Une condition peut être construite à l'aide :

- **d'opérateurs de comparaison :** >, <, <=, =>, == ou ! =.
- **d'opérateurs logiques :** `and`, `or`, `not`.

Une **instruction conditionnelle** est une instruction qui n'est exécutée que **si** une condition est réalisée. La condition est suivie de **deux points**. Les instructions liées à la condition doivent être **indentées**.

IV.2 Structures conditionnelles

Définition 2

- La structure `if... else` permet d'exécuter des instructions **si** une condition est réalisée et d'autres instructions **sinon**.
  Exemple :

```
a = 4
if 6*a-4>1:
    x = 3
else :
    x=4

print(x)
```

- La structure `if... elif... else` permet d'utiliser plusieurs conditions qui s'excluent l'une l'autre.Il est possible d'utiliser autant de `elif` qu'on le souhaite.
  Exemple :

```
a = -2
if 6*a-4 == 0:
    x = 3
elif 6*a-4>0:
    x = 2
else:
    x = 4

print(x)
```

Exercice 4 ★☆☆

Créer une fonction qui prend en paramètre `x` et renvoie sa valeur absolue. La première ligne sera donc :

```
def MaFonctionValeurAbsolue(x):
```

♥ On n'oublie pas de **tester** sa fonction en l'appelant après sa définition avec plusieurs valeurs possibles!

Exercice 5 ★☆☆

Créer une fonction à un paramètre entier `n` qui renvoie `True` si `n` est pair et `False` si `n` est impair. Pour savoir si `n` est pair ou impair, utiliser le reste de la division euclidienne de `n` par 2. La première ligne sera donc :

```
def SuisJePair(n):
```

Vocabulaire

`True` et `False` sont ce qu'on appelle des **booléens**. C'est un certain type de variable en Python. Les booléens correspondent au *Vrai* et *Faux* du cours de logique.

Exercice 6 ★☆☆

On considère une équation du second degré $ax^2+bx+c=0$.

1. Écrire une fonction, qui prend en argument des paramètres `a`, `b` et `c`, et qui renvoie le nombre de solutions de l'équation du second degré. La première ligne sera donc

```
def MaFonctionTrinome(a,b,c):
```

2. Modifier cette fonction, pour qu'elle renvoie aussi les éventuelles solutions.
   Indication : `3**0.5` renvoie la racine carré de 3, si jamais vous avez deux solutions `x` et `y`, vous tapez donc un `return 2,x,y`
   Tester avec `a=1`, `b=0` et `c=1` ainsi qu'avec `a=1`, `b=0` et `c=-9`.

3. Évidemment si `a=0` ce n'est pas une équation du second degré. Modifier le programme pour en tenir compte. ‡

‡. *Never trust user input* : dans un programme, il faut essayer de parer les bugs qui pourraient être causer par des valeurs absurdes rentrées par un utilisateur

V Boucles bornées et non bornées

Définition 3

Les boucles en informatique permettent de **répéter une instruction autant de fois que nécessaire**. Il y a deux types de boucles en informatique :

- les boucles **bornées**, ou boucles *for*, où l'on précise dès le début combien de fois il faut répéter l'opération,
- et les boucles **non bornées**, ou boucles *while*, où l'on donne une condition d'arrêt.

V.1 Prérequis : la commande `range`

Définition 4

Pour a et b deux entiers, `range(a,b)` permet d'obtenir l'ensemble ordonné des nombres entre a et $b-1$. `range(b)` est un raccourci désignant `range(0,b)`. On peut aussi utiliser `range(a,b,c)` pour préciser le pas (l'écart entre les valeurs).

V.2 Boucles bornées : la boucle *for*

Définition 5

La boucle *for* permet de répéter une instruction pour chaque valeur d'une variable muette évoluant dans une liste de nombres prédéfinie.

Recopier et exécuter les lignes suivantes :

```
44 for i in range(10):
45     print("Ligne ",i," : en CPGE, je travaille régulièrement.")
```

Il faut faire attention à la première et à la dernière valeur prises par la boucle. C'est un piège classique.

Exercice 7 ★☆☆

Une boucle **for** permet, par exemple, de calculer des sommes : supposons que l'on veuille calculer $S = 0+1+2+3+\ldots+n$, Pour cela, définir une variable `S` à 0, créer une boucle où `i` prendra toutes les valeurs de 0 à n, à chaque étape, `S` devient `S+i`. Écrire ce script, avec une variable `n` puis tester pour $n = 100$, est-ce cohérent avec votre raisonnement mathématique?

Exercice 8 ★☆☆

Considérons une suite $(u_n)_{n\in\mathbb{N}}$ telle que $u_0 = 2$ et pour tout $n \in \mathbb{N}$, $u_n = 2\times u_{n-1}+3$.

1. Écrire un script, permettant de calculer u_{100}.
   Indication : les deux derniers chiffres de u_{100} sont 7 et 7.
2. Adapter ce qui précède pour écrire une fonction ayant pour paramètre `n` renvoyant la valeur de u_n. Ainsi la première ligne sera :

```
46 def MaSuite(n):
```

Remarque 2. Les mots clefs sont `for` et `in`. Comme en mathématique, la variable muette n'a pas forcément à intervenir dans les commandes à répéter. On attire l'attention sur les deux choses suivantes :

- **Il ne faut pas oublier** : dans le code.
- **L'indentation est primordiale.** C'est elle qui signifie à Python comment sont organisés les blocs de codes à exécuter et leur agencements entre eux, à la manière des parenthèses pour organiser un calcul en mathématiques.

V.3 Boucles non bornées : la boucle *while*

Définition 6

Avec une boucle *while*, on répète la ou les instructions associées tant qu'une **condition d'arrêt** n'est pas vérifiée.

Prenons, par exemple, un nombre `x`, et supposons qu'on le divise par 5 tant que `x` est supérieur ou égale à 5. Contrairement à une boucle for, ici on n'a aucune idée de combien de fois on va diviser `x` par 5. Cela donne le code suivant :

```
x=182
while x>=5:
    x=x/5
print(x)#Comme on a fini la boucle while, x<5
```

Exercice 9 ★☆☆

Rajouter, dans le code précédent, une variable `compteur` qui compte le nombre de fois où on a divisé par 5.

Attention aux boucles infinies ! La boucle *while* continue tant que la condition est vérifiée. Aussi, il faut être sûr qu'il n'y a qu'un nombre fini d'étapes. Sinon, vous êtes condamnées à attendre une éternité, ou, dans la pratique, à faire surchauffer votre machine qui ne répondra plus.
Ne tentez donc pas d'écrire un code comme celui-ci :

```
x = 182
while x >= 5 and x != 8:
    x = x+1
```

Exercice 10 ★★☆

1. Créer une fonction `S`, qui a un entier naturel `n` renvoie la moitié de `n` si `n` est pair et `3n+1` sinon.
2. Partant de $n = 10$, calculer, à la main, $S(n)$, puis $S(S(n))$ puis $S(S(S(n)))$ etc. Que constatez-vous ?
3. Soit `N` une variable entière et positive. Écrire un script qui transforme `N` en $S(N)$ tant que `N` est différent de 4, 2 et 1. Afficher au fur et à mesure les valeurs prises par la variable `N`.
4. Tester pour différentes valeurs de `N`.
5. Modifier le script pour compter combien il faut d'étapes pour arriver à 4, 2 ou 1.
6. Parmi tous les nombres entre trois et un million, quel(s) est(sont) le(les) nombre(s) où il a fallu le plus d'étapes ? Quel est ce nombre d'étapes maximum ?

La conjecture de Syracuse est l'hypothèse que pour tout entier $n \geq 2$, on finit par arriver à 4, 2 ou 1. À ce jour, personne n'a jamais réussi à le démontrer.

Exercice 11 ★☆☆

Créer une fonction qui dépend de deux paramètres `a` et `b` (deux entiers positifs et $b \neq 0$). et qui renvoie `q` et `r` le quotient et le reste de la division euclidienne de `a` par `b`.
On n'utilisera évidemment pas les commandes % et \\. À la place, on retirera `b` à `a` jusqu'à obtenir un reste plus petit que `b`.

VI Listes

Définition 7

Une **liste** est une collection **ordonnée** d'éléments. On peut coder des listes en Python en utilisant les crochets. Leur type est : `list`.

Nous détaillons peu les listes ici, car nous y reviendrons dans un futur TP. Voici toutefois quelques commandes sur les listes :

- `L[k]` renvoie le k-ième élément de la liste `L`.
- `len(L)` renvoie le nombre d'éléments de la liste `L`.
- `L.append(a)` rajoute l'élément `a` à la fin de la liste `L`.

Remarque 3. La commande `range` renvoie en réalité une liste. Les listes `range` sont fréquemment utilisées et ont un type spécialisé : le type `range`.

Taper le code suivant :

```
L = [-10,5,-10,8]
print(L[0])
print(L[1])
print(L[2])
print(L[3])
print(L[4])#Provoque une erreur ! Pourquoi ?
print(L[-1])
L[2] = 23#Modifie la valeur du troisième élément de L
print(L)
print(len(L))#Affiche le nombre d'éléments
```

La numérotation commence à 0! `L[1]` ne renvoie pas la valeur du premier élément de la liste mais du deuxième!
Dans le jargon, on dit que «*array starts at* 0». [a]

a. Selon le langage de programmation utilisé, les listes sont initialisées soit à 0, soit à 1.

Exercice 12 ★☆☆

Écrire une fonction prenant en argument une liste et qui, à l'aide d'une boucle *for*, **affiche** tous les éléments de cette liste. *Voici un exemple de fonction qui ne renvoie rien!*

Exercice 13 ★☆☆

Écrire une fonction prenant en argument une liste et qui, à l'aide d'une boucle *for*, renvoie la somme de tous les éléments de cette liste.
On pourra s'aider d'une variable `S` à qui on ajoute à l'étape `k` de la boucle le `k`-ième élément de la liste.

Exercice 14 ★★★

Un triplet pythagoricien est la donnée de trois entiers naturels non nuls `(a,b,c)` avec $c^2 = a^2 + b^2$.

1. Donner un exemple d'un tel triplet.
2. Écrire un script, qui compte combien il y a de tels triplets `(a,b,c)` avec `a`, `b` et `c` inférieurs ou égals à 1000 et avec $a \leq b$.
3. Modifier le script à la question précédente, pour avoir la liste de tous ces triplets. *On pourra s'inspirer des lignes suivantes :*

```
64 L = []#Crée une liste vide
65 T = (4,5,6)#Crée un triplet
66 L.append(T)#Rajoute à la liste L le triplet T
```