1 Introduction

Au cours des années de CPGE, vous allez apprendre un langage informatique nommé Python. Pour manipuler Python, on utilisera un IDE [1] nommé Pyzo ou Sypder. Le but étant que vous ayez des bases solides en algorithme.

2 Variables

Une variable permet de stocker une information. 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 caractères
print(s)
```

On peut évidemment effectuer les opérations usuelles sur les variables qui sont des nombres [2] :

```
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 [3].

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 [4] les valeurs de a et b. Tester si cela fonctionne.

3 Fonctions

Comme en mathématiques, une fonction en Python, renvoie un résultat qui dépend des paramètres donnés :

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

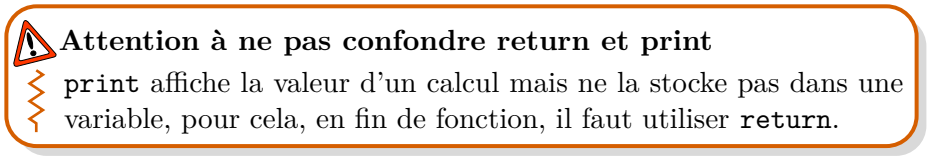
Exercice 3 (⋆)**.** Taper ces lignes et exécutez. Que se passe-t'il ? Essayez maintenant la commande suivante et indiquer ce que calcule la fonction mystère :

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

Notez que pour qu'une fonction fonctionne, plusieurs choses sont nécessaires :

1. Pour «Integrated Development Environment» : Environnement de développement
2. 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.
3. Certains langages notent `a←b` pour souligner que la variable `a` va changer de valeur et prendre celle de `b`.
4. La commande `a,b=b,a` est bien évidemment interdite.

- Sur la première ligne, le mot clé `def` pour dire que l'on définit une fonction, le nom de la fonction, entre parenthèse le ou les paramètres de la fonction [5], les deux points `:` à la fin de la ligne
- Les instructions qui sont dans la fonction doivent subir une indentation.
- Enfin, le `return` permet de retourner le résultat obtenu [6].



4 Conditions

Les conditions permettent d'effectuer certaines instructions uniquement dans certains cas. Comment écrire ces cas ? Par exemple, si a et b sont des nombres, on effectuer une instruction que si $a > b$ ou si $a \geqslant b$ ou encore si $a = b$. Sous Python, cela se passe ainsi :

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

Recopier ce code, et exécutez le plusieurs fois, en faisant changer la valeur de la variable `age`. Ne pas oublier les deux points. De même, il faut indenter les instructions à réaliser lorsque la condition est vérifiée

5. Il est possible de n'avoir aucun paramètre, dans ce cas on laisse les parenthèses sans rien dedans `def MaFonction():`

6. Il est possible de ne pas avoir de return, mais dans ce cas, on ne pourra pas sauvegarder le résultat de la fonction.

Exercice 4 ($\star$)**.** Créer une fonction qui prend en paramètre `x` et renvoie sa valeur absolue. La première ligne sera donc :

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

Ne pas oublier de tester cette fonction avec plusieurs valeurs possibles.

Exercice 5 ($\star$)**.** Créer une fonction à un paramètre entier `n` qui renvoie [7] `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):
```

Exercice 6 ($\star$)**.** On considère une équation du second degré $ax^2+bx+c = 0$.

1. Écrire une fonction, qui dépend 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 [8]

5 Les boucles

5.1 Boucle for

La boucle `for` répète un certain nombre de fois une instruction avec une variable partant généralement à 0 et s'arrêtant à une valeur fixé en avance. Recopier et exécuter les lignes suivantes :

7. `True` et `False` sont ce qu'on appelle des booléens : ils correspondent au Vrai et Faux du cours de logique.

8. Never trust user input : dans un programme, essayez de parer les bugs qui pourraient être causer par des valeurs absurdes rentrées par un utilisateur.

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

Péril imminent : à la dernière valeur prise par la boucle

La variable `i` est donc partie de 0 et s'est arrêtée à 9 (et non 10) !!!!

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 vos maths ?

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 :

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

5.2 Boucle while

Une boucle while continue tant qu'une condition est réalisé. 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.

Par contre, ne tentez pas d'écrire un code comme celui-ci :

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

Péril imminent : 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 condamnés à attendre une éternité [a].

a. Et comme dit le proverbe *«l'éternité c'est long, surtout vers la fin !»*.

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 $n \geqslant 2$, on finit par arriver à 4, 2 ou 1. Cependant, personne n'a jamais réussi à le montrer[9]*. Un moyen pour vous de devenir riche et célèbre ?*

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`.

9. Idriss Aberkane prétend l'avoir démontré, mais aucun chercheur en mathématiques ne pense que sa preuve est sérieuse.

6 Listes

Une liste contient des éléments, on accède à ces éléments grâce à leur numéro dans la liste. Tapez 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
```

Attention : la numérotation commence à 0

`L[1]` ne renvoie pas la valeur du premier élément de la liste mais du deuxième !

Exercice 12 (⋆)**.** Écrire une fonction, qui dépend d'un paramètre - une liste - et qui, à l'aide d'une boucle for, affiche tous les éléments de la liste [10].

Exercice 13 (𝄞⋆)**.** Écrire une fonction, qui dépend d'un paramètre - une liste - et qui, à l'aide d'une boucle for, renvoie la somme de tous les éléments de la 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 pythagorien 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 $a \leqslant b$.
3. Modifier le script à la question précédente, pour avoir la liste de tous ces triplets. *On pourra s'inspirer des lignes suivantes :*

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

10. C'est un exemple de fonction qui ne comporte pas de **return**.