Série I : Premiers pas en Python 3

But : Cette série a pour but de permettre une première prise de contact avec l'environnement informatique du collège et faire les premiers pas avec le langage Python.

Exercice I.0 : L'environnement informatique.

- Utiliser Python en mode interactif : Ouvrir une fenêtre en mode **interactif** (Chercher Python depuis Démarrer, ouvrir « IDLE Python *etc*» que l'ordinateur va proposer.
  Essayer d'écrire >>> 1 + 1 Python sait calculer ! On verra qu'il est capable de bien plus, heureusement ! Le mode interactif permet de rapidement tester une commande. Alors n'hésitez pas à essayer de votre propre chef d'autres commandes pour vous assurer de son rôle ou pour votre simple curiosité.

- Sauvegarder des documents: nous verrons plus tard qu'il est possible d'écrire votre script dans un éditeur de texte, puis de dire à Python de l'exécuter. Cette façon comporte de nombreux avantages comme la sauvegarde de vos documents et l'écriture de longs scripts. D'une manière générale, prenez l'habitude lorsque vous sauvegardez vos documents de les classer rigoureusement en utilisant des sous-dossiers. Évitez les caractères non ASCII (accents, cédilles . . .). Il est inutile de sauver sur l'ordinateur de la salle informatique : les données élèves y sont régulièrement effacées (p. ex. au démarrage). Il est risqué de sauver sur un support amovible (absence de backup). Votre compte élève sur le serveur de l'école est vraiment le bon emplacement !

Exercice I.1 : C'est parti !

Jouons un peu avec Python avec le mode interactif en testant les commandes suivantes et essayez d'en tirer des conclusions.

REMARQUE I.1.A : les espaces ne sont pas pris en considération en Python.

```
>>> 2-9
>>> 7 + 3 * 4
>>> (7 + 3) * 4
>>> 20 / 3
>>> 20 // 3
>>> 18 / 3
```

```
>>> 18 // 3
>>> 20 % 3
>>> 18 % 3
>>> 4 * 4 * 4
>>> 4 ** 3
```

```
>>> 5 ** 1 * 2
>>> 5 ** (1 * 2)
>>> 5 * 1 ** 3
>>> (5 * 1) ** 3
>>> 59 * 100 // 60
>>> 59 * (100 // 60)
```

Que représentent les symboles **, // et % ?
Et maintenant :

```
>>> 20.0%3
```

Pour comprendre la différence avec 20%3 testons la commande : **`type(valeur)`**

```
>>> type(20)
>>> type(20.0)
>>> type(20%3)
>>> type(20.0%3)
```

```
>>> 20,0
>>> type(20,0)
>>> type((20,0))
```

Que signifie **`type()`** ?

Ce cours provient de M. Cédric Paychère.
Quelques modifications mineures, de style et de mise en page ont été effectuées.

Exercice I.2 : **Quels types ?**

L'exercice précédent montre qu'il existe plusieurs types. A vous d'en découvrir quelques autres avec leurs propriétés.

```
>>> type(123)
>>> type(123456789012345678)
>>> type(123456789012345678.)
>>> 123456789012345678 - 123456789012345677
>>> 123456789012345678. -123456789012345677.
```

Et si on mélange ces types ?

```
>>> 20.0 / 3
```

Nous verrons encore d'autres types par la suite du cours lorsque nous en aurons besoin.

Exercice I.3 : **Stockons !**

Il est possible d'affecter à une variable une valeur. L'opération d'affectation est représentée par le symbole =

```
>>> a = 12
>>> b = 2
>>> a
>>> c = a - b
>>> x = y = 7          Spécificité Python
>>> x
>>> y
>>> a, b = 4, 8        Spécificité Python
>>> a
>>> b
```

REMARQUE I.3.A : **Mots réservés**

Attention il existe des noms de variables réservés, comme `if` ou `and`
En effet ces noms sont utilisés par le langage lui-même.

```
>>> if = 24
>>> fi = 24
```

Il y a 33 mots réservés :

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

Nous apprendrons par la suite leurs propriétés. Encore quelques mises en garde :

```
>>> Coucou = 11
>>> coucou = 13
>>> Coucou
>>> coucou
>>> coucou1 = 14
>>> coucou% = 13
>>> 1coucou = 15
>>> maVariable = 16
```

Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans maVariable, par exemple.

Exercice I.4 : **Affichons !**

Il existe deux manières d'afficher la valeur d'une variable. La première consiste simplement à entrer

au clavier le nom de la variable comme vous l'avez déjà constaté. Cependant lorsque vous n'utiliserez plus le mode interactif cette fonctionnalité ne sera plus possible. C'est pourquoi à l'intérieur d'un programme, il faudra utiliser la commande **`print(variable)`**.

```
>>> essai = 2
>>> print(essai)
```

Exercice I.5 : **À vous de deviner !**

Décrivez le plus clairement et le plus complètement possible ce qui se passe à chacune des sept lignes de l'exemple ci-dessous, puis testez votre réponse en écrivant le script.

```
>>> anneeCourante = 2017
>>> anneeNaissance = 2000
>>> nom = 'Billy'
>>> age = anneeCourante - anneeNaissance
>>> print(nom, age)
```

Quel est le type de nom ?

Exercice I.6 : **Composons.**

Python est un langage de programmation évolué. Il est possible de construire des instructions complexes par assemblage de fragments divers. Ainsi par exemple, si vous savez comment additionner deux nombres et comment a cher une valeur, vous pouvez combiner ces deux instructions en une seule. Voici quelques exemples :

```
>>> print(17 + 3)
>>> anneeCourante, anneeNaissance, nom = 2017, 2000, 'Billy'
>>> print( nom, 'a un age de ', anneeCourante - anneeNaissance)
```

REMARQUE I.6.A : la composition **doit permettre de rendre un code clair et concis. Si une composition est trop longue, elle risque d'altérer sa lisibilité.**

REMARQUE I.6.B : **Pour ceux qui souhaitent installer Python chez eux.**

Les concepteurs du langage ont choisi de rompre la compatibilité entre les versions v2 et v3 du langage. Ces deux versions ont leurs propres vies avec leurs révisions, il y a donc en quelque sorte deux langages Python. Comme nous travaillons avec la révision 3 je vous la recommande. Vous avez la possibilité de télécharger gratuitement Python chez vous sur le site officiel : http://www.python.org/download/.

Une fois l'installation terminée, un éditeur de commande sera par la même occasion disponible : il se nomme IDLE (Python GUI). Cet éditeur vous sera nécessaire pour exécuter vos scripts sans passer par le mode interactif, et surtout pour enregistrer vos programmes! Remarque : il existe d'autres éditeurs de commandes. Pour n'en citer que quelques-uns: SciTE, Smultron, Editra ou Nedit. En fait, n'importe quel éditeur de texte permet l'écriture d'un script, que vous pourrez ensuite sauvegarder, modifier, copier, etc. comme n'importe quel autre texte traité par ordinateur. L'avantage d'un éditeur « intelligent » est d'adapter sa coloration syntaxique lorsque le nom du script se termine par l'extension .py

Par la suite, il suffira d'entrer la commande python3 MonScript.py dans une fenêtre de terminal pour l'exécuter.

La version 3 du langage apporte quelques changements de fond qui lui confèrent une plus grande cohérence et même une plus grande facilité d'utilisation, mais qui imposent une petite mise à jour de tous les scripts écrits pour les versions précédentes. Des outils de conversion existent pour analyser des scripts développés pour une version antérieure, (voir en particulier le script **2to3.py**), maintenu en ligne sur le site web http://inforef.be/swi/python.htm.

Exercice I.7 : **Type(s), priorité et mathématique*.**

Assignez les valeurs respectives 3, 5, 7 à trois variables *a*, *b*, c. Effectuez l'opération a – b/c. Quels sont les types des variables et du résultat ? Le résultat est-il mathématiquement correct ? Si ce n'est pas le cas, comment devez-vous procéder pour qu'il le soit ?

Exercice I.8 : **Calcul d'aire***

Ecrivez des instructions de sorte à pouvoir calculer l'aire d'un disque de rayon r.

Exercice I.9 : **Assez dormi ?***

Ecrivez des instructions de sorte à pouvoir calculer le nombre d'heures et de minutes de sommeil étant donné l'heure à laquelle vous vous êtes couchés, et l'heure à laquelle vous vous êtes levés.

REMARQUE I.9.A : **Bibliographie**

La documentation Python est abondante, en général pragmatique et souvent teintée d'humour britannique (style Monty Python bien sûr), n'hésitez pas à chercher sur internet. Pour mon cours, j'ai utilisé les sources suivantes :

Sources de mon cours Python :

- A disposition au CEDOC :Gérard Swinnen, ***Apprendre à programmer avec Python 3***, ouvrage distribué suivant les termes de la Licence Creative Commons « Paternité-Pas d'Utilisation Commerciale-Partage des Conditions Initiales à l'Identique - 2.0 France ».
- www.mathex.net (2016)
- Eric Von Aarburg, mon collègue de Calvin
- F. Marchino, Programmation Python 16-17, Collège et Ecole de Commerce Emilie-Gourd

* La présence d'une étoile après un exercice indique qu'un corrigé écrit de l'exercice viendra.

Corrigé Premiers pas en Python 3

Solution Exercice I.7 :

```
1   a = 3 #on initialise trois variables (en type int)
2   b = 5
3   c = 7
4   #Python 3 comprendra implicitement que le type du résultat en float
5   resultat = a−b/c
6
7   print (resultat)
8
9
10  #Pour un affichage plus elegant − nous verrons ceci dans peu de temps .
11  print ( ”Le resultat de ” + str(a) + ”−” + str(b) + ”/” + str(c) + ” est de ” + str(resultat))
```

Solution Exercice I.8 :

```
1   r = 8
2   pi = 3.14159
3   aire = pi * r**2
4
5   print (aire)
6
7
8   #Pour un plus joli texte :
9   print (”L’aire du disque de rayon” + str(r) + ” est de ” + str(aire))
```

Solution Exercice I.9 :

```
1
2   # Dans cet exemple on se leve a 7h02 . . .
3   hLever = 7 #heure du lever
4   mLever = 02 #minutes du lever
5
6   # . . . et on se couche a 22h34
7   hCoucher = 22 #heure du coucher
8   mCoucher = 28 #minutes du coucher
9
10  minutesAvantMinuit = (23 − hCoucher ) * 60 + (60−mCoucher ) #lenb de minutes de sommeil avant
    minuit
11  minutesApresMinuit = 60 * hLever + mLever #lenb de minutes de sommeil apres minuit
12  totalMinutesSommeil = minutesAvantMinuit + minutesApresMinuit #lenb de minutes total de
    sommeil
13
14  heuresSommeil = totalMinutesSommeil / 60 #conversion du nb de minutes total en heure
15  heuresSommeil = (heuresSommeil % 24) #pour ne pas dormir p lus de 24 heures
16  minutesSommeil = totalMinutesSommei l % 60 #et en minutes
17
18  print ( heuresSommeil , minutesSommeil )
19
20
21  #pour l’affichage plus style
22  print (str(heuresSommeil) + ” heures et ” + str(minutesSommeil) + ” minutes ” )
```

Série II : Les boucles et les conditions I

But : Le but de cette série est d'utiliser l'éditeur de commandes pour écrire des programmes un peu plus complexes faisant notamment intervenir les notions de boucles et de conditions.

Exercice II.0 : Utilisons Python IDLE ou GUI (Graphical User Interface)

Dans la série précédente, tous vos scripts ont été exécutés dans le mode interactif. Voyons maintenant comment exécuter vos commandes par le logiciel Python IDLE. Ouvrez ce dernier et écrivez dans une fenêtre:

```
nom='Albertlevert'
print("Voici le premier programme de " + nom )
```

Remarquez que, en écrivant dans un éditeur de commande, des couleurs sont mises automatiquement pour faciliter la lecture de votre code. Sauvegardez ces lignes sur le disque dur de votre ordinateur ou dans votre espace élève (E:/) ou sur votre clef USB dans un dossier /informatique/serie2/ préalablement créé. Le nom du fichier doit avoir l'extension .py. Par exemple vous pouvez enregistrer ce petit bout de code sous le nom exII1.py. Exécutez ce dernier et vérifiez son bon fonctionnement.

Exercice II.1 : L'interrogatoire bugge!

Ecrivez les commandes suivantes dans un fichier « exII2interrogatoire.py » et exécutez-les (RUN ou F5).

```
nom = input("Comment vous appelez-vous ? ")
print("Bonjour " + nom)
anneeNaissance = input("Quelle est votre année de naissance ? ")
age = 2017 - anneeNaissance
print("Bonjour " + nom + ", selon nos renseignements vous avez " + str(age) + " ans. ")
```

Il y a un bug… Déterminez l'action de la fonction input() et de la fonction str(). Puis essayez de corriger l'erreur. L'exercice qui suit peut vous aider.

Exercice II.2 : Convertissons les types.

Essayez également dans le mode interactif de déterminer le rôle des fonctions `str()`, `int()` , `float()` en écrivant par exemple les lignes suivantes :

```
>>> int(3.6)
>>> float(3)
>>> str(3)
>>> 2**1000
>>> float(2**100)
>>> int('3')
```

Utilisez la fonction `type()` si vous avez un doute !

Exercice II.3 : **Égalité surprenante ?**

Si vous essayez de résoudre l'équation mathématique : `x = x + 1` vous n'allez pas trouver de solution! Mais le signe = est d'une autre signification en informatique. Toujours dans le mode interactif stockez un nombre dans la variable x, puis incrémentez-le de un.

```
>>> x = 10
>>> x = x + 1
>>> print(x)
>>> x = x + 1
>>> print(x)
>>> x = x + 1
>>> print(x)
```

Exercice II.4 : **L'instruction `while`.**

Déterminer l'action de l'instruction `while` en exécutant le script suivant:

```
1  n=0
2  while n <10 :
3      print(n)
4      n=n+1
```

Que se passe-t-il si on efface la ligne 4 ? Et si l'on supprime l'indentation ? Donnez une explication !

REMARQUE II.4.A : **Python tient compte de la mise en page**

Ce qui précède nous permet de faire le point sur quelques règles de syntaxe :

- Une **ligne d'instruction** se termine par le caractère (invisible dans les traitements de texte) de retour à la ligne ou par le caractère `#` (qui indique le début d'un commentaire). L'interpréteur ignore tout ce qui suit `#`.
- Un **bloc d'instructions** débute avec une ligne d'en-tête contenant une instruction bien spécifique (`if, elif, else, while, def`, etc.) se terminant par un double point.
- Les blocs sont délimités par **l'indentation** : toutes les lignes d'un même bloc doivent être indentées exactement de la même manière (c'est-à-dire décalées vers la droite d'un même nombre d'espaces). Le nombre d'espaces à utiliser pour l'indentation est quelconque, mais la plupart des programmeurs utilisent des multiples de 4. Le code du premier bloc ne peut pas être écarté de la marge. Attention l'utilisation de tabulation (autorisée) est peu recommandée : espaces et tabulations sont des codes binaires distincts : Python considérera donc que des lignes indentées différemment font partie de blocs différents, même si visuellement le résultat parait identique à l'écran. Il peut en résulter des erreurs difficiles à déboguer.

Exercice II.5 : **Les opérateurs `==` ; `!=` ; `>` et `<`**

Déterminez le rôle des opérateurs `==` ; `!=` ; `>` et `<` en essayant dans le mode interactif :

```
>>> 2 > 1
>>> 2 < 1
>>> 1 == 1
>>> 1 == 2
>>> 2 != 1
>>> 2 != 2
```

Exercice II.6 : **L'instruction `if`, `elif` et `else`.**

Déterminez l'action des instructions `if , elif` et `else` en exécutant le script suivant:

```
n=int(input('Tapez un nombre: '))
if n>100:
    print("Ce nombre est plus grand que 100. " )
elif n<10:
    print("Ce nombre est plus petit que 10. " )
else:
    print("Ce nombre est compris entre 10 et 100. " )
```

Exercice II.7 : **Punition ringarde***

Écrivez un programme qui écrit 100 fois le texte : "Je dois écrire 100 fois un texte inutile."

Exercice II.8 : **Cherche mon nombre***

Avec les instructions while et if programmez un jeu où le joueur doit deviner un nombre compris entre 1 et 99 en faisant des essais. À chaque essai, le programme donne l'indication si le nombre est plus grand ou plus petit en fonction du nombre proposé par le joueur.

Exercice II.9 : **Table de multiplication***

Avec l'instruction while écrivez un programme qui demande à l'utilisateur un nombre puis affiche la table de multiplications de ce nombre de 1 à 10.

REMARQUE II.9.A : **La fonction input() renvoie toujours une chaîne de caractères.**

Le bug de l'Exercice II.1 : provient de la fonction input() qui renvoie toujours une chaîne de caractères. **Si l'on veut que l'entrée soit un nombre, il faut le convertir.**

```
ch = input("Veuillez entrer un nombre entier positif quelconque : ")
nn = int(ch)      #conversion de nn en nombre de type integer
print("Le carré de ", nn, "vaut", nn**2)

a = input("Entrez une donnée numérique : ")
print("Le type de a est", type(a))
b = float(a)      #conversion de a en nombre de type float
print("Le type de b est", type(b))
```

REMARQUE II.9.B : **La fonction `print()`**

Pour afficher la réponse à Exercice II.1 :, nous avons concaténé des chaînes de caractères à l'aide du + (texte1+texte2+texte3) et appelé la fonction `print` de cette unique chaîne. Dans la remarque ci-dessus nous voyons que la fonction `print` affiche différents arguments séparés par des virgules. Le résultat à l'écran est identique. Mais l'usage d'arguments permet de configurer individuellement chaque élément de la chaîne (couleurs, nombre de décimales, etc.)

Corrigé Les boucles et les conditions I

Solution Exercice II.7 :

```
1 punitionRingarde = " Je dois écrire 100 fois un texte inutile "
2 n = 0
3
4 while n < 100 :
5     print ( punitionRingarde )
6     n = n + 1
```

Solution Exercice II.9

```
 1 print ("Le nombre a trouvé est composé de 2 chiffres" )
 2 print (" " )
 3 nbSecret = 79
 4 nbEssai = 0
 5 while nbEssai != nbSecret :
 6   nbEssai = int(input ("Quel est mon nombre ? " ))
 7   if nbEssai > nbSecret :
 8     print ("mon nombre est plus petit ! " )
 9   if nbEssai < nbSecret :
10     print ("mon nombre est plus grand ! " )
11
12 print ("Bravo ! Tu as trouvé mon nombre ! " )
```

Solution Exercice II.9 :

```
 1 nb = int(input ("Quel livret ? " ))
 2
 3 a = 1
 4 while a < 11:
 5   print ( str(nb) + " x " + str(a) + " = " + str(nb * a ) )
 6   a = a + 1
```

Série III : Les boucles et les conditions II

But : Le but de cette série est d'approfondir les notions de boucles et de conditions vues la semaine précédente.

Conseils d'apprentissage :

a) Avant de commencer une nouvelle série, il est vivement conseillé de regarder les corrigés de la dernière série. En effet, comparer vos codes avec d'autres peut donner une autre approche à la résolution d'un exercice ou une manière plus "élégante" de raisonner.
b) Je vous conseille également de rédiger au fur à mesure votre glossaire contenant chaque nouvelle commande que vous apprenez.

Exercice III.0 : L'instruction `break`

Déterminer l'action de l'instruction `break` en exécutant le script suivant:

```
      print ('Devinez le nombre qui se cache . . . ' )
      while (1 < 2):
      nn = int(input("Tapez un chiffre : "))
      if (nn == 3):
    print ("OUI" )
    break
      else:
    print ("NON" )
```

Exercice III.1 : Tant que nous y sommes, continuons avec les tant que (`while`) !

Essayez de comprendre le code suivant en décomposant pas à pas ce que Python devrait exécuter. Ensuite, vérifiez en l'exécutant. Surpris ?

```
aa = 1
bb = 1
while (aa < 12):
  print("\nla table de " , aa)  # \n correspond à un retour à la ligne
  while (bb < 12):
    print(aa , "x" , bb , "=" , aa*bb)
    bb += 1 # Ajoute 1 à bb
  bb = 1
  aa += 1 # Ajoute 1 à aa
```

Exercice III.2 : Tant que, tant que, tant que.

Dans la même logique que l'exercice précédent, le programme suivant teste tous les nombres de 1 à 999 pour découvrir le code secret d'un cadenas à chiffres.

```
nbSecret = 659
aa = bb = cc = 0

while (aa < 10):
  while (bb < 10):
    while (cc < 10):
      nbGenerer = 100 * aa + 10 * bb + cc
      #print(aa, bb, cc)
      if (nbGenerer == nbSecret):
        print("Bravo, vous avez réussi trouver le mot de passe ! ")
        print( nbGenerer )
      cc = cc + 1
    cc = 0
    bb += 1 # Ajoute 1 à bb
  bb = 0
  aa += 1
```

Essayez d'enlever le commentaire # de la ligne "`print(aa, bb, cc)`" .

Exercice III.3 : **Viète***

Résolution d'une équation quadratique. Créez un programme qui demande à l'utilisateur les coefficients *a*, *b* et *c*, puis affiche les solutions (si elles existent...) de l'équation quadratique: $a \cdot x^2 + b \cdot x + c = 0$.

Exercice III.4 : **Jeu de Nim***

Le jeu des allumettes : ce jeu se joue à deux joueurs. Au départ 30 allumettes sont disposées. Puis, à tour de rôle, chaque joueur enlève au maximum 3 allumettes. Le joueur qui prend la dernière allumette a perdu ! Exemple d'affichage:

```
Quel est ton nom joueur 1 ?Eric
Quel est ton nom joueur 2 ?Pere Noel
-----------------------------------------------
 Information : Il reste 30 allumettes.
-----------------------------------------------
Combien enlèves-tu  d'allumettes Eric ?2
-----------------------------------------------
 Information : Il reste 28 allumettes.
-----------------------------------------------
Combien enlèves-tu  d'allumettes Pere Noel ?1
-----------------------------------------------
 Information : Il reste 27 allumettes.
-----------------------------------------------
Combien enlèves-tu  d'allumettes Eric ?4
Tu ne peux pas enlever plus que 3 allumettes !
Combien enlèves-tu  d'allumettes Eric ?
```

Exercice III.5 : **Cherche mon nombre, je t'aide***

Variante du jeu du "cherche mon nombre" :
Avec les instructions `while`, `if`, `elif` et `break`, programmez un jeu où le joueur doit deviner un nombre en faisant des essais. À chaque essai, le programme donne une réponse plus ou moins encourageante (glacé, froid, tiède, chaud, brûlant) en fonction de l'erreur commise par le joueur. Au final, le nombre d'essais mis par le joueur doit-être affiché avec un petit commentaire.

Bonus : le joueur gagne un diamant de taille égale à son nombre d'essais !

```
    #
   ###
  #####
 #######
#########
 #######
  #####
   ###
    #
```

Corrigé Les boucles et les conditions II

Solution Exercice III.3 :

```
print( ”résolveur d'équations du 2ème degré : a x^2 + b x + c = 0 ” )

aa = float( input( "que vaut le coefficient a ?" ) )
bb = float( input( "que vaut le coefficient b ?" ) )
cc = float( input( "que vaut le coefficient c ?" ) )

delta = bb**2 – 4*aa*cc

if (delta < 0):
   print( "Il n’existe pas de solution réelle ! " )

elif (delta == 0):
   xx = –bb / (2* aa )
   print( "La solution est : " + str(xx) )

else:
   x1 = (–bb + delta **0.5) / (2 * aa )
   x2 = (–bb – delta **0.5) / (2 * aa )
   print( "Les solutions sont : " + str( x1) + " et " + str( x2) )
```

Solution Exercice III.4 :

```
# ex_III05a_Jeu_de_Nim.py
'''
Jeu de Nim,
Il y a deux joueurs.
Un tas commence avec 30 allumettes.
Chacun à leur tour, un joueur doit enlever 1, 2 ou 3 allumettes du tas.
Celui qui pend la dernière allumette a perdu.
La stratégie gagnante est simple.
Une variante est d'interdire de prendre le même nombre d'allumettes
que le joueur précédent.
'''
nb_allumettes = 30

joueur1 = input( ' Quel est ton nom joueur 1 ? ' )
joueur2 = input( ' Quel est ton nom joueur 2 ? ' )

while ( nb_allumettes > 0) :
  print ( '-----------------------------------------------' )
  print ( ' Information : Il reste ', nb_allumettes, 'allumettes.' )
  print ( '-----------------------------------------------' )

  nb_enleve = 4
  while( (nb_enleve > 3) or (nb_enleve < 1) or (nb_enleve > nb_allumettes)):
    try:
      nb_enleve = int(input( "Combien enleves-tu d’allumettes " + joueur1 + " ? " ))
      if ( nb_enleve > 3):
        print ('Tu ne peux pas enlever plus que 3 allumettes ! ' )
      elif ( nb_enleve < 1):
        print ('Il faut enlever au moins une allumette ! ' )
      elif ( nb_enleve > nb_allumettes ):
        print ("Tu ne peux pas enlever autant d’allumettes ! " )
      else:
        nb_allumettes = nb_allumettes - nb_enleve
        joueur1, joueur2 = joueur2, joueur1 # échange le nom des joueurs.

      if ( nb_allumettes == 0 ):
        print ( "******************** \n \n \n \n " )
        print ( joueur1, ' a gagné ! ', joueur2, ' a perdu, car il a pris la dernière allumette.')
        break

      if ( nb_allumettes == 1 ):
        print ( "******************** \n \n \n \n " )
        print ( joueur2, ' a gagné ! ', joueur1, ' doit prendre la dernière allumette ! ! ! ')
        nb_allumettes = 0 # assure de sortir de la boucle principale
        break

    except:
      print("Entre un chiffre entre 1 et 3")
```

Solution Exercice III.5 :

```
# ex_III06a_Devine_mon_nombre.py
'''
Un joueur doit deviner le nombre caché par l'ordinateur.
Des indications lui sont données.
Lorsqu'il a trouvé, un joli dessin lui est présenté.
Utilise la librairie "random"
'''
import random

nbCache = random.randint(10, 99)
print("J'ai caché un nombre entre 10 et 99, essaye de le deviner")

nbTentatives = 0

while True:
  nbTentatives += 1
  try:
    nbEssai = int(input("Tapez un nombre entre 10 et 99 : "))

    if (nbEssai == nbCache):
      print ("Bravo, tu as trouvé en", nbTentatives, "tentatives.")
      break
    elif abs( nbEssai - nbCache ) <= 2:
      print("C’est bouillant !")

    elif abs( nbEssai - nbCache ) <= 5:
      print("C’est chaud !")

    elif abs( nbEssai - nbCache ) <= 10:
      print("C’est tiede !")

    elif abs( nbEssai - nbCache ) <= 20:
      print ("C’est froid !")

    else:
      print ("C’est glacé !")
  except:
    print("Entrez un nombre entre 10 et 99")

print( "Vous avez trouvé le nombre caché en ", nbTentatives, "tentatives !" )

if (nbTentatives < 2):
  print( "Sacré coup de chance ! ! !")
elif (nbTentatives < 5):
  print ( "Joli score !")
elif nbTentatives < 8 :
  print ("Le score est acceptable.")
else :
  print( "pas terrible comme score ! ! !")
```

Solution avec le Bonus:

```
print() # pour sauter une ligne

ligne = 0
largeurMax = 4

while (ligne < largeurMax):
  largeur = 1* ( ( largeurMax ) - ligne )
  nbDiese = 2* ligne + 1
  print ( largeur * ' ' + nbDiese * '#' )
  ligne = ligne + 1

while (ligne >= 0):
  largeur = 1* ( ( largeurMax ) - ligne )
  nbDiese = 2* ligne + 1
  print ( largeur * ' ' + nbDiese * '#' )
  ligne = ligne - 1
```

Série IV : Boucles et conditions III - Nombres aléatoires

But : Le but de cette série est d'approfondir encore les notions de boucles et de conditions à travers deux problèmes et un jeu.

Exercice IV.0 : Simulation des rebonds d'une balle*

L'objectif de cet exercice est de résoudre le problème suivant : lorsqu'une balle tombe d'une hauteur initiale h, sa vitesse à l'arrivée au sol est de $v = \sqrt{2 \cdot g \cdot h}$ avec g = 9,81 [m/s²]. Immédiatement après le rebond, sa vitesse est $v_1 = e \cdot v$, où e est une constante et v la vitesse avant le rebond.

Elle remonte alors à la hauteur $h_1 = \frac{v_1^2}{2 \cdot g}$. Le but est d'écrire un programme "rebonds.py" qui calcule la hauteur à laquelle la balle remonte après un nombre `nbr` de rebonds.

Méthode : On veut résoudre ce problème, non pas du point de vue formel (avec des équations) mais par simulation du système physique (la balle). Utilisez l'instruction `while` et des variables `v, v1` (les vitesses respectivement avant et après le rebond), et `h, h1` (les hauteurs au début de la chute et à la fin de la remontée).

Tâches : Écrivez le programme "rebonds.py" qui affiche la hauteur après le nombre de rebonds spécifiés. Votre programme devra utiliser la constante g, de valeur 9, 81 [m/s²] et demander à l'utilisateur d'entrer les valeurs de :

- h0 (hauteur initiale, contrainte : h0 ≥ 0) ;
- e (coefficient de rebond, contrainte 0 ≤ e < 1) ;
- nbr (nombre de rebonds, contrainte : 0 ≤ nbr).

Essayez les valeurs h0 = 25 [m], e = 0,9 et nbr = 10. La hauteur obtenue devrait être environ 3,04 [m].

Exercice IV.1 : Nombres premiers*

Il existe de nombreuses applications industrielles sur la notion de nombres premiers, par exemple dans certains systèmes cryptographiques. L'exercice suivant propose une méthode pour déterminer si un nombre est premier ou non. Écrivez le programme "premier.py" qui demande à l'utilisateur d'entrer un entier n strictement plus grand que 1, puis décide si ce nombre est premier ou non.

Algorithme :

- vérifier si le nombre n est pair (si oui, il n'est pas premier sauf si c'est 2).
- pour tous les nombres impairs inférieurs ou égaux à la racine carrée de n, vérifier s'ils divisent n. Si ce n'est pas le cas, alors n est premier.

Rappel : Un nombre n est divisible par D si le reste de la division de n par D est nul.

Tâches : Si n n'est pas premier, votre programme devra afficher le message: "Le nombre n'est pas premier, car il est divisible par D", où D est un diviseur de n autre que 1 et n. Sinon, il devra afficher le message: "Ce nombre est premier".

Testez votre programme avec les nombres : 2, 16, 17, 91, 589, 1001, 1009, 1299827 et 2146654199. Indiquez ceux qui sont premiers.

Si vous voulez devenir riche et célèbre sachez que "Electronic Frontier Foundation" offre 100'000 dollars pour la découverte d'un nombre premier d'au moins 10 millions de chiffres décimaux !

REMARQUE IV.1.A : **Générateur aléatoire de nombres**

Certains problèmes sont difficiles (voir impossibles) à résoudre de manière formelle (avec des équations). La simulation est alors souvent utilisée comme pour résoudre le premier exercice. Nous verrons durant l'année d'autres problèmes où la simulation permettra de résoudre des problèmes complexes. Générer des nombres aléatoires permet justement de simuler une expérience, et ainsi 1 de tester théoriquement un cas "réel". Par exemple, au lieu d'analyser l'influence de l'ouverture d'un guichet de poste supplémentaire : coût, temps d'attente, taux d'occupation, etc avant même son installation, il suffira de simuler aléatoirement (selon une certaine loi de distribution) le va-et-vient des personnes, ce qui ne prendra que quelques centièmes de secondes au programme. Selon le résultat de la simulation, les responsables pourront prendre la décision d'ouvrir ou non ce guichet supplémentaire.

Il existe un paquet sur Python pour générer des nombres aléatoirement. Pour charger cette librairie il suffit d'ajouter dans l'entête du programme la ligne suivante :
`from` `random` `import *`.
Puis à l'intérieur de votre programme pour générer un nombre aléatoire uniforme entre a et b, il faudra écrire la commande `randint(a,b)`. Ci-dessous un petit exemple :

```
from random import *
a=1
b=10
nbAleatoire = randint (1 , 10)
print("Nbr aleatoire entre " + str(a) + " et " + str(b) + " : " +
str(nbAleatoire))
```

Exercice IV.2 : **Jeu du mémory***

À chaque tour du jeu "memory", votre programme doit générer un chiffre aléatoire entre 1 et 4. Le joueur doit se souvenir à chaque tour de tous les chiffres tirés précédemment. C'est-à-dire votre programme doit demander au joueur, après avoir généré un nouveau chiffre, d'écrire tous les anciens nombres ainsi que le dernier. Si ce dernier nombre (composé des bons chiffres) est correct alors votre programme fonctionne.

Exercice IV.3 : **Jeu du Blackjack****

La partie oppose la banque (l'ordinateur) à un joueur. Le but est d'approcher ou de faire un total de 21 sans les dépasser. La valeur des cartes est comptée de la manière suivante :
— de 2 à 10 : la valeur nominale de la carte ;
— chaque figure : 10 points ;
— l'As : 1 ou 11 points (à choix du joueur).
Le joueur mise une somme d'argent au début du jeu. Le croupier (l'ordinateur) tire deux cartes (deux nombres aléatoires compris entre 1 et 13) : une pour lui et l'autre pour le joueur. Si le nombre aléatoire généré est 11, 12 ou 13 cela correspond respectivement au Valet, à la Dame ou au Roi. Si la carte tirée est l'As, le croupier demande au joueur si sa valeur est de 1 ou 11 points. A chaque tour, le croupier demande au joueur s'il veut encore une carte, si oui un nouveau nombre est généré sinon le joueur s'arrête. Le croupier, lui, adopte toujours la même règle : "La banque tire à 16, s'arrête à 17". Au final, les totaux du croupier et du joueur sont comparés. Si le joueur a un total supérieur au croupier, alors le joueur double sa mise de départ, sinon il la perd. Celui qui dépasse 21 perd sa mise.

Corrigé Boucles et conditions III - Nombres aléatoires

Solution Exercice IV.0 :

```
# ex_IV01a_Rebonds_ball.py
'''
Simulation de rebonds d'une balle, avec perte d'énergie à chaque rebond.
Avec gestion d'erreur d'entrée : try: ... except: ...
'''
while True :
    h0 = float(input( 'Quelle est la hauteur initiale en [m] ? ' ))
    if (h0 >= 0):
        break

while True :
    eps = float(input( 'Quel est le coefficient de rebond ? ' ))
    if (eps >= 0) and (eps < 1):
        break

while True :
    try:
        nbr = int(input( 'Combien de rebonds de balle ? ' ))
        if (nbr < 0):
            print('Un nombre positif de rebonds :', end=' ')
        else:
            break
    except ValueError:
        print('Un nombre entier de rebonds :', end=' ')

rebonds = 0
g = 9.81
h = h0

while (rebonds < nbr):
    print('rebond', rebonds,' ', h)
    v = (2 * g * h) ** 0.5
    v1 = eps * v
    h = (v1 **2) / (2 * g )
    rebonds = rebonds + 1

print( 'Après', rebonds, 'rebonds la balle atteint une hauteur de', h, 'mètres.')
```

Solution Exercice IV.1 :

```
# ex_IV02a_Test_nombre_premier.py
'''
Test si un nombre donné par l'utilisateur est un nombre premier.
'''
while True :
    nn = int(input( 'Entrez un nombre entier supérieur à 1 : '))
    if (nn > 1):
        break

isPremier = True
if (nn % 2 == 0) and (nn != 2):
    print( 'Le nombre n'est pas premier, car il est divisible par 2 !')
    isPremier = False

mm = 3 # diviseurs possibles du nombre nn
while (mm <= nn**0.5):  # On teste avec tous les nombres <= racine carrée de nn.
    if (nn % mm == 0):
        print ('Le nombre', nn, "n'est pas premier, car il est divisible par", mm, '!')
        isPremier = False
        break
    mm = mm + 2 # essaye la division avec le prochain nombre impair.

if (isPremier):
    print( 'Le nombre', nn, 'est premier !')
```

Solution Exercice IV.2 :

```
# ex_IV03a_Jeu_de_memory.py
'''
Un joueur doit mémoriser une suite de nombre et les écrire ensuite.
Utilise la librairie random.
'''
from random import *

c = n = 0

while True:
    a = randint(1 , 4 )
    print (100 * '\n' + 'Le nouveau nombre est ' + str( a ) )
    c = a + c * 10
    b = int(input( '\nRedis tous mes nombres : ' ))
    if (b != c):
        print( "FAUX, c'était          :", c )
        break
    n = n + 1

print( 'Nb de réussites :', n)
```

Exemple de fonctionnement:

```
Le nouveau nombre est 3
Redis tous mes nombres :3
Le nouveau nombre est 2
Redis tous mes nombres :32
Le nouveau nombre est 1
Redis tous mes nombres :321
Le nouveau nombre est 1
Redis tous mes nombres :3212
FAUX !
Nb de réussites : 3
```

**Blackjack

Série V : Données composites - Chaînes de texte

But : Le but de cette série est de travailler avec un premier type de donnée composite : les chaînes de caractères.

Exercice V.0 : **Testons et comprenons ce que font les instructions suivantes :**

```
>>> print("ah! "*100)
>>> print(10*"\n")
>>> chaine="Christine"
>>> chaine
>>> len(chaine)
>>> chaine[0:3]
>>> len(chaine[0:3])
>>> chaine[2:7]
>>> chaine[2:]
>>> chaine[2:-2]
>>> chaine[-2:2]
```

REMARQUE V.0.A : **Caractères spéciaux avec la fonction `print()`**

Nous avons déjà utilisé la fonction `print()` qui permet d'afficher des chaînes de caractères, l'antislash (\) permet quelques subtilités complémentaires :

- l'antislash permet d'écrire sur plusieurs lignes un contenu qui serait trop long pour tenir sur une seule (cela vaut pour n'importe quel type d'instruction) (essayer dans shell) :
  salut = "Ceci est une chaîne plutôt longue contenant \
  plusieurs lignes de texte"
- À l'intérieur d'une chaîne de caractères, l'antislash permet d'insérer un certain nombre de codes spéciaux (sauts à la ligne \n, apostrophes \', guillemets \", etc.).

Le *triple quotes* (""" ou ''') permet aussi d'insérer des caractères « exotiques » (y compris l'antislash lui-même) dans une chaîne.

Exercice V.1 : **Afficher des guillemets et des apostrophes***

Réécrivez, de quatre façons différentes, le programme :

```
phrase1 = 'les œufs durs.'
phrase2 = '"Oui", répondit-il,'
phrase3 = "j'aime bien"
print(phrase2, phrase3, phrase1)
```

à chaque fois à l'aide d'une seule fonction `print(???)`, sans variable, qui affichera strictement le même texte (en respectant la présence de guillemets et d'apostrophe) que l'exemple.

Exercice V.2 : **Accès aux caractères individuels d'une chaîne**

Les chaînes de caractères constituent un cas simple d'un type de données plus général que l'on appelle des données composites : elles ne sont composées que des caractères eux-mêmes. Il est possible de considérer la chaîne de caractères comme un seul objet ou comme une collection de caractères distincts disposés dans un certain ordre. Par conséquent, chaque caractère de la chaîne peut être désigné par sa place dans la séquence, à l'aide d'un index.

Exemple :

```
>>> ch = "Christine"
>>> print(ch[0], ch[5], ch[8])
C t e
```

REMARQUE V.2.A : **Attention !**

- la chaîne est numérotée à partir de zéro (1er caractère ch[0])
- le dernier caractère (ch[8]) est donc d'une unité de moins que la longueur de la chaîne :
```
>>> len(ch)
9
```
- Il n'est pas possible de modifier 1 caractère (l'instruction `ch[2]="R"` est refusée), la parade : constituer une chaîne modifiée.
```
>>>                                  #ch[:2] du début ([0]) à la position 2 non comprise
ch=ch[:2]+"R"+ch[2+1:]               #puis on ajoute R, qui sera donc en position 2
>>> ch                               #ch[3:] de la position 3 à la fin (len(ch))
'ChRistine'
```

Exercice V.3 : **Chercher un caractère dans une chaîne***

Écrivez un script avec une boucle `while` qui détermine si une chaîne contient ou non le caractère « e ».

Exercice V.4 : **Instruction for**

Le parcours d'une séquence est une opération très fréquente en programmation. Pour en faciliter l'écriture, Python vous propose une structure de boucle plus appropriée que la boucle `while`, basée sur le couple d'instructions : "`for ... in ...`" :
Avec for, le programme ci-dessus devient :

```
chaine = input ("Taper une chaine de caractères, je vous dirai s'il y a un e\n")

for lettre in chaine:
    if lettre=="e":
        break

if lettre=="e":
    print("Il y a un e")
else:
    print("Je n'ai pas vu de e")
```

Exercice V.5 : **Compter les apparitions d'un caractère dans une chaîne***

Écrivez un script, en utilisant une boucle for, qui compte le nombre d'occurrences du caractère « e » dans une chaîne.

Exercice V.6 : **Concaténer***

Écrivez un script (ou plutôt 2, l'un avec while, l'autre avec for) qui recopie une chaîne (dans une nouvelle variable), en insérant des astérisques entre les caractères.
Ainsi par exemple, « gaston » devra devenir « g*a*s*t*o*n »

Exercice V.7 : **Inverser***

Écrivez un script (ou plutôt 2, l'un avec while, l'autre avec for) qui recopie une chaîne (dans une nouvelle variable) en l'inversant.
Ainsi par exemple, « zorglub » deviendra « bulgroz ».

Exercice V.8 : **Palindrome***

D'après l'exercice précédent, écrivez 2 scripts qui déterminent si une chaîne de caractères donnée est un palindrome (c'est-à-dire une chaîne qui peut se lire indifféremment dans les deux sens), comme dans « radar » ou « kayak ».

Corrigé Données composites - Chaînes de texte

Solutions Exercice V.1 :

```
print('"Oui", répondit-il, j\'aime bien les œufs durs')
```

ou

```
print("\"Oui\", répondit-il, j'aime bien les œufs durs")
```

ou

```
print("""Oui", répondit-il, j'aime bien les œufs durs""")
```

ou

```
print('''"Oui", répondit-il, j'aime bien les œufs durs''')
```

Solution Exercice V.3 :

```
# ex_V04a_Cherche_un_caractere.py
'''
Compte le nombre de "e" dans une chaîne de caractères
'''
chaine = input("Taper une chaine de caractères, \
je vous dirai le nombre de 'e' qu'elle contient.\n")

lesE="eéèêëEÈË" #Astuce : on va tester différents "e", on les groupe dans une chaîne

nCompte = 0 # Compte le nombre de 'e' dans la chaîne
nn = 0
while (nn < len(chaine)):
  if (chaine[nn] in lesE):
    #  in   permet de vérifier si un élément fait partie d'une séquence
    nCompte += 1
  nn += 1

if (nCompte > 0):
    print("Il y a", nCompte, "caractères 'e' dans la chaîne :", chaine)
else:
    print("Il n'y a pas de 'e' dans la chaine :", chaine)
```

Solution Exercice V.5 :

Avec WHILE

```
chaine = input ("Taper une chaîne de caractères, je
vous dirai combien il y a de e\n")
etre
n=nbE=0
while n < len(chaine):
   if chaine[n]=="e":
      nbE+=1
   n+=1

print("Il y a "+str(nbE)+" e")
```

Avec FOR

```
chaine = input ("Taper une chaine de caractères, je
vous dirai combien il y a de e\n")

nbE=0
for car in chaine:
   if car=="e":
      nbE+=1


print("Il y a "+str(nbE)+" e")
```

Solution Exercice V.6 :

```
chaine = input ("Donnez-moi une chaine que j'y
insère des * !\n")

chaineEtoile=""
n=0
while n<len(chaine):
   chaineEtoile+=chaine[n]+"*"
   n+=1
print(chaineEtoile)
```

```
chaine = input ("Donnez-moi une chaine que j'y
insère des * !\n")

chaineEtoile=""
for car in chaine:
   chaineEtoile+=car+"*"


print(chaineEtoile)
```

Solution Exercice V.7 :

```
chaine = input ("Donnez-moi une chaine à
inverser !\n")

chaineInverse=""
n=len(chaine)
while n>0:
   n-=1
   chaineInverse+=chaine[n]
print(chaineInverse)
```

```
chaine = input ("Donnez-moi une chaine à
inverser !\n")

chaineInverse=""
for car in chaine:
   chaineInverse=car+chaineInverse


print(chaineInverse)
```

Solution Exercice V.8 :

```
chaine = input ("Donnez-moi une chaine à
inverser !\n")

chaineInverse=""
n=len(chaine)
while n>0:
   n-=1
   chaineInverse+=chaine[n]

if chaineInverse==chaine:
   print(chaine+" est un palindrome")
else:
   print(chaine+" n'est pas un palindrome")
```

```
chaine = input ("Donnez-moi une chaine à
inverser !\n")

chaineInverse=""
for car in chaine:
   chaineInverse=car+chaineInverse


if chaineInverse==chaine:
   print(chaine+" est un palindrome")
else:
   print(chaine+" n'est pas un palindrome")
```

Série VI : Les fonctions

But : Créer des fonctions personnelles afin de simplifier la lecture d'un script voire de simplifier un problème complexe en le décomposant en sous-problèmes simples.

Exercice VI.0 : **Testons :**

Exécutez le script suivant et concluez :

```
def livretDe7():
    nn=1
    while nn<11:
        print(nn*7,end=" ")
        nn+=1
    print()

livretDe7()
```

Et pour celui-là essayez pour différentes valeurs :

```
def livretDe(base):
    n=1
    while n<11:
        print(n*base,end=" ")
        n+=1
    print()

while True:
    try:
        a=int(input("Quelle base ? (Taper Q pour quitter) : "))
    except:
        break
    livretDe(a)
```

REMARQUE VI.0.A : **Utilisation d'argument**

Dans le premier exemple, la fonction `livretDe7()` n'a pas d'argument – ses parenthèses restent vides – la fonction exécute toujours le même chose. Dans le second exemple, la fonction est généralisée, son résultat dépend d'un argument, en l'occurrence une variable. Il est possible d'avoir plusieurs arguments, dans ce cas il faut appeler la fonction avec tous ses arguments dans le bon ordre.

Exercice VI.1 : **Fonction avec plusieurs paramètres***

Définissez une fonction `tableMulti()` qui améliore la fonction `livretDe()` de sorte que l'on puisse choisir – grâce à 3 arguments – non seulement la base, mais aussi le début et la fin du fragment de la table de multiplication à afficher.

Que se passe-t-il lors de l'exécution de ce script ?

```
t, d, f = 11, 5, 10
while t<21:
    tableMulti(t,d,f)
    print()
    t, d, f = t +1, d +3, f +5
```

Exercice VI.2 : **Les paramètres facultatifs d'une fonction**

Exécuter le script suivant et déterminer l'action de chaque instruction.

```
def essai ( x, y=0) :
    return(x+y)
print(essai( 21 ))
print(essai ( 21, y=2))
```

Exercice VI.3 : **Testons une variable « rr » est-elle la même que dans la boucle principale ?**

Premier exemple (créer un fichier et l'exécuter) :

```
#rr = 130
def chezMoi():
    pp=100
    rr=200
    print("chezMoi:", end=" ")
    print(pp, end=" ")
    print(qq, end=" ")
    print(rr)

# boucle principale
pp, qq = 110, 120
chezMoi()
print("dehors :", end=" ")
print(pp, end=" ")
print(qq, end=" ")
print(rr)
```

Deuxième exemple (modifier le fichier précédent et l'exécuter) :

```
#rr = 130
def chezMoi():
    global pp, rr #ajout au premier exemple
    pp=100
    rr=200
    print("chezMoi:", end=" ")
    print(pp, end=" ")
    print(qq, end=" ")
    print(rr)

pp, qq = 110, 120
chezMoi()
print("dehors :", end=" ")
print(pp, end=" ")
print(qq, end=" ")
print(rr)
```

REMARQUE VI.3.A : **Portée des variables**

Dans le premier script, les variables pp, qq et rr sont utilisées à deux endroits différents (dans la fonction "chezMoi" ou dans la boucle principale). Afin d'éviter des conflits Python distingue les variables locales (utilisées pour les besoins internes d'une fonction) de celles globales (utilisées dans le programme général).

- Python crée deux variables locales (pp et rr) visibles uniquement à l'intérieur de la fonction où elles sont modifiées. Même si ailleurs des variables portent le même nom il s'agit de variables distinctes ;
- Python ne crée pas de variable locale pour qq, puisque que qq n'est pas modifiée par la fonction. La valeur de qq est donc la même dans la fonction et à l'extérieur, cela semble pratique, mais **c'est dangereux** : une fonction devrait être complètement autonome pour rester générale, pour une programmation propre il faudrait passer qq en paramètres. Même s'il n'est pas recommandé d'utiliser des variables globales, l'instruction « global » permet d'avoir accès partout dans le code (y compris en modification) à une variable.
- Quant à rr, elle n'est utilisée que dans la fonction et n'est donc pas visible à l'extérieur, ce qui provoque l'erreur lors du print(rr), puisqu'elle est mise en commentaire dans la première ligne.

Dans le deuxième script, pp et rr sont définies comme globales (ce qui est peu recommandable), elles sont visibles et modifiables partout.

Exercice VI.4 : `Return`

Comme en math, les fonctions peuvent aussi retourner une valeur :

```
def cube(x):
    return x**3
```

Ainsi `a=cube(2)` affectera la valeur 8 à la variable `a`.
NB : Formellement, le vocabulaire informatique distingue les fonctions (lorsqu'un résultat est retourné) des procédures (qui ne renvoient rien).

Exercice VI.5 : **Fonction retournant une chaîne***

Définissez une fonction ligneCar(nn, car) qui renvoie une chaîne de n caractères car.

Exercice VI.6 : **Fonction retournant un livret dans une chaîne***

Reprendre la fonction livretDe(base) de sorte quelle retourne une chaîne de caractères affichables avec l'instruction print(livretDe(?))

Exercice VI.7 : **Fonction retournant un réel***

Définissez une fonction surfCercle(rr). Cette fonction doit renvoyer la surface (l'aire) d'un cercle dont on lui a fourni le rayon rr en argument. Par exemple, l'exécution de l'instruction :
print(surfCercle(2.5)) doit donner le résultat : 19.63495...

Exercice VI.8 : **Fonction maximum***

Définissez une fonction maximum(n1,n2,n3) (essayez c'est la fonction max(), mais n'utilisez pas max() pour cet exercice) qui renvoie le plus grand de 3 nombres n1, n2, n3 fournis en arguments. Par exemple, l'exécution de l'instruction :
print(maximum(2,5,4)) doit donner le résultat : 5.

Exercice VI.9 : **Compter une occurrence**

Définissez une fonction compteCar(car,ch) (essayez la méthode ch.count(), mais ne l'utilisez pas pour cet exercice) qui renvoie le nombre de fois que l'on rencontre le caractère car dans la chaîne de caractères ch. Par exemple, l'exécution de l'instruction :
print(compteCar('e', 'Cette phrase est un exemple')) doit donner le résultat : 7

Exercice VI.10 : **Docstrings**

Reprenons le script de la série qui inverse les l'ordre des caractères d'une chaîne quelconque et définissons la fonction inverse(ch) :

```
def inverse(ch):
    """La chaine inversée est renvoyée au programme appelant."""
    chaineInverse=""
    for car in ch:
        chaineInverse=car+chaineInverse
    return(chaineInverse)

print(inverse(input ("Donnez-moi une chaine à inverser ! ")))
```

Exécutez une fois ce script, puis tapez dans shell :

```
>>> help(inverse)
```

Que voyez-vous apparaître ?

En PYTHON, il est recommandé d'insérer un commentaire, juste sous la définition d'une fonction ou procédure, décrivant son utilisation et son action. Si vous tapez help(fonction) c'est ce commentaire (docstring) que vous verrez apparaître. Pratique, n'est-ce pas ?

Exercice VI.11 : **Compter** les mots* hkjhdfas klfhas fhkas dfk kdsfhdkshfksdfhkdshfasdh

Définissez une fonction compteMots(ph) qui renvoie le nombre de mots contenus dans la phrase ph. On considère comme mots les ensembles de caractères (sauf les espaces) inclus entre des espaces.

Exercice VI.12 : **Les nombres à deviner**

a) Calculez la somme suivant : $\frac{4}{1}-\frac{4}{3}+\frac{4}{5}-\frac{4}{7}+\frac{4}{9}-\frac{4}{11}+\ldots\pm\frac{1}{(2*N+1)}$, avec $N = 10000$

Si N tends vers l'infini, vers quelle nombre tend cette somme ?

b) Définissez la fonction : $f(x)=\frac{2x+2}{2x+1}\cdot\frac{2x+2}{2x+3}$

Calculez le produit : $f(0)*f(1)*f(2)*f(3)*f(4)*\ldots*f(N)$, avec N raisonnablement grand.

Si N tends vers l'infini, vers quelle nombre tend ce produit ?

Exercice VI.13 : **Suites de Syracuse I***

Exercice VI.14 : **Suites de Syracuse II**

Exercice VI.15 : **Paramètres par défaut**

Définissez la fonction politesse (avec 2 arguments) ci-dessous

```
def politesse(nom, vedette = "Monsieur") :
      print("Veuillez agréer, ", vedette, nom, "mes cordiales salutations")
```

Et appelez-la
(avec 1 argument)

```
>>> politesse('Dupont')
```

(avec 2 arguments)

```
>>> politesse('Schmitt', 'Mlle')
```

Que se passe-t-il ?

Exercice VI.16 : **Paramètres dans le désordre**

Définissez la fonction oiseau ci-dessous

```
def oiseau(voltage = 100, etat = 'allumé', action = 'danser la java') :
      print("Ce perroquet ne pourra pas", action)
      print("si vous le branchez sur", voltage, "volts !")
      print("L'auteur de ceci est complètement", etat)
```

Et expliquez les exécutions suivantes :

```
>>> oiseau(etat='givré', voltage=250, action='vous approuver')
>>> oiseau()
```

REMARQUE VI.16.A : **Paramètres par défaut**

Une fonction peut avoir des valeurs par défaut (vedette = « Monsieur ») pour tout ou partie des paramètres. Il est alors possible d'appeler la fonction avec une partie seulement des arguments attendus.
Attention : dans la définition de la fonction, les paramètres sans valeur par défaut doivent précéder les autres !
La définition ci-dessous serait incorrecte : def politesse(vedette = "Monsieur", nom) :

Lors de l'appel de la fonction, les arguments utilisés doivent être fournis dans le même ordre que celui des paramètres correspondants… Sauf (comme dans l'appel de a fonction oiseau) dans le cas où tous les paramètres ont reçu une valeur par défaut. Mais alors, il faut désigner nommément les paramètres !

Corrigé
Les fonctions

Corrigé Fonction avec plusieurs paramètres* Exercice VI.1 :

```
def tableMulti(base, debut, fin):
    nn=debut
    while (nn<fin):
        print(nn*base, end=" ")
        nn+=1
```

Corrigé Fonction retournant une chaîne* Exercice VI.5 :

```
def ligneCar(nn, car):
    chaine=""
    for ii in range(0, nn):
        chaine += car
    return chaine

def hmi_ligneCar():
    lettre=input("Quelle lettre ? ")
    nb=int(input("Combien de fois la répéter ? (<=0 pour quitter) : "))
    if (nb<=0):
        return False
    print(ligneCar(nb,lettre))
    return True

while hmi_ligneCar():
    continue
```

Corrigé Fonction retournant un livret dans une chaîne* Exercice VI.6 :

```
def livretDe(base):
    nn = 1
    resultat=""
    while (nn<11):
        resultat = resultat + " " + str(nn*base)
        nn += 1
    return resultat

while True:
    try:
        aa = int(input("Quelle base ? (Taper une lettre pour quitter) : "))
    except:
        break
    print(livretDe(aa))
```

Corrigé Fonction retournant un réel* Exercice VI.7 :

```
def surfCercle(rr):
    from math import pi
    return pi*rr**2

def hmi_surfCercle():
    rayon = float(input("Quel est le rayon du cercle ? (<=0 pour quitter) : "))
    if (rayon <= 0):
        return False
    print(surfCercle(rayon))
    return True

while hmi_surfCercle():
    continue
```

Corrigé Fonction maximum* Exercice VI.8 :

```
def maxi(a, b):
    if a>b:
        return a
    else:
        return b

def maximum(n1, n2, n3):
    return maxi(maxi(n1, n2), n3)

def hmi_maximum():
    try:
        a=float(input("Entrez un premier nombre : (Q pour quitter)"))
    except:
        return False
    b=float(input("Entrez un deuxième nombre :"))
    c=float(input("Entrez un deuxième nombre :"))
    print(maximum(a, b, c))
    return True

while hmi_maximum():
    continue
```

Corrigé Compter les mots* Exercice VI.11 :

```
def compteMot(chaine):
    """Renvoie le nombre de mots de la phrase
(un mot etant tout ensemble de caracteres
inclus entre des espaces)"""
    nbMot=0
    separateur=True
    for car in chaine:
        if car==" ":
            separateur=True
        elif separateur:
            nbMot+=1
            separateur=False
        else:
            separateur=False
    return nbMot

def hmi_compteMot():
    phrase=input("Entrez une phrase : (QQ pour quiter)")
    if phrase=="QQ":
        return False
    print("Il y a", compteMot(phrase)," mots dans", phrase)
    return True

while hmi_compteMot():
    continue
```

Corrigé Les nombres à deviner Exercice VI.12 :

```
def calcule(k):
    return (float(2*k + 2) / (2*k + 1) * (2*k + 2) / (2*k + 3))

n = 0
produit = 1

while n < 100000:
    produit = produit * calcule(n)
    n = n + 1

print(2*produit)
```

Corrigé Suites de Syracuse I Exercice VI.13 :

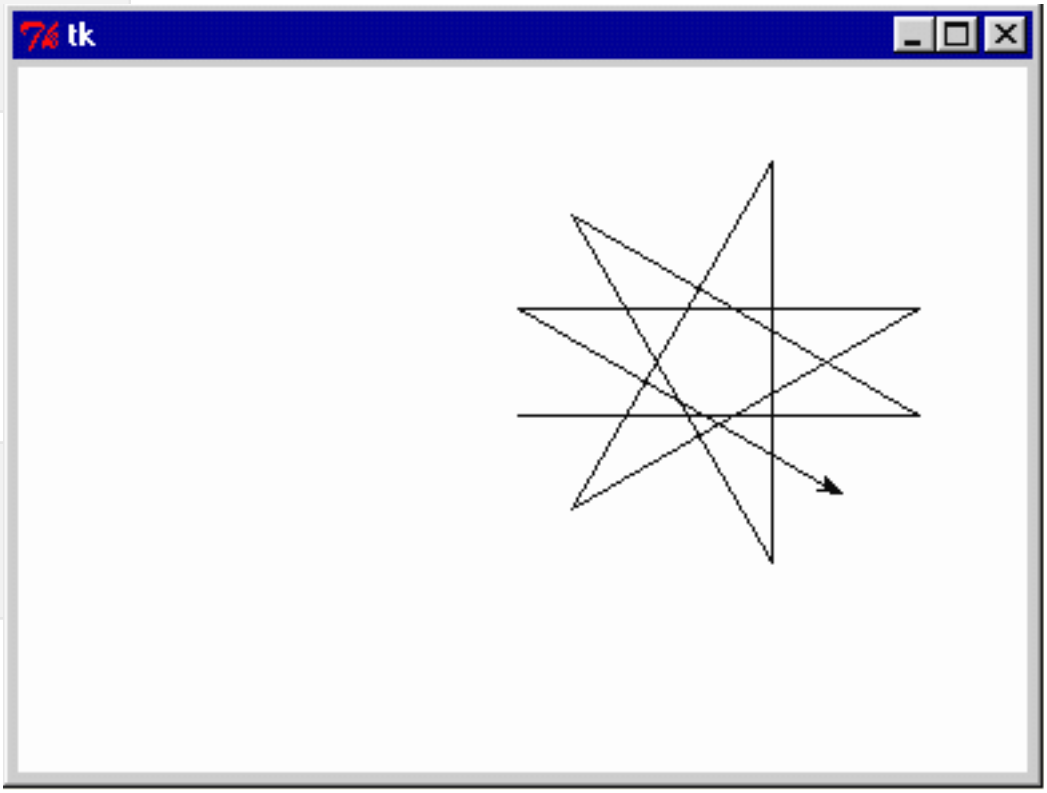
```
from turtle import *
setworldcoordinates(0,0,20,200)
speed(0)

def f(n):
        if n%2==0:
                a=n/2
        else:
                a=3*n+1
        return a
print("Tapez un nombre entier >0: "),
n=int(input())
print("u_0  = ",n)
k=0
while n>1:
        k=k+1
        n=f(n)
        print("u_",k," = ",n)

        goto(k,n)
done()
```

Série VII : Le module turtle

L'une des grandes qualités de Python est qu'il est extrêmement facile de lui ajouter de nombreuses fonctionnalités par importation de divers *modules*.

Pour illustrer cela, et nous amuser un peu avec d'autres objets que des nombres, nous allons explorer un module Python qui permet de réaliser des « graphiques tortue », c'est-à-dire des dessins géométriques correspondant à la piste laissée derrière elle par une petite « tortue » virtuelle, dont nous contrôlons les déplacements sur l'écran de l'ordinateur à l'aide d'instructions simples. Essayons tout de suite :

```
>>> from turtle import *
>>> forward(120)
>>> left(90)
>>> color('red')
>>> forward(80)
```



L'exercice est évidemment plus riche si l'on utilise des boucles :

```
>>> reset()
>>> a = 0
>>> while a <12:
a = a +1
forward(150)
left(150)
```

Amusez-vous à écrire des scripts qui réalisent des dessins suivant un modèle imposé à l'avance. Les principales fonctions mises à votre disposition dans le module *turtle* sont les suivantes :

reset() On efface tout et on recommence
goto(x, y) Aller à l'endroit de coordonnées x, y
forward(distance) Avancer d'une distance donnée
backward(distance) Reculer
up() Relever le crayon (pour pouvoir avancer sans dessiner)
down() Abaisser le crayon (pour recommencer à dessiner)
color(couleur) *couleur* peut être une chaîne prédéfinie ('red', 'blue', etc.)
left(angle) Tourner à gauche d'un angle donné (exprimé en degrés)
right(angle) Tourner à droite
width(épaisseur) Choisir l'épaisseur du tracé
fill(1) Remplir un contour fermé à l'aide de la couleur sélectionnée
write(texte) *texte* doit être une chaîne de caractères

Exercice VII.0 : **Expérience**

Importez le module turtle (`from turtle import *`) pour pouvoir effectuer des dessins simples.
Vous allez dessiner une série de triangles équilatéraux de différentes couleurs.
Pour ce faire, définissez d'abord une fonction triangle() capable de dessiner un triangle d'une couleur bien déterminée . (`def triangle(couleur):`)
Utilisez ensuite cette fonction pour reproduire ce même triangle en différents endroits, en changeant de couleur à chaque fois.

Exercice VII.1 : **Devinette**

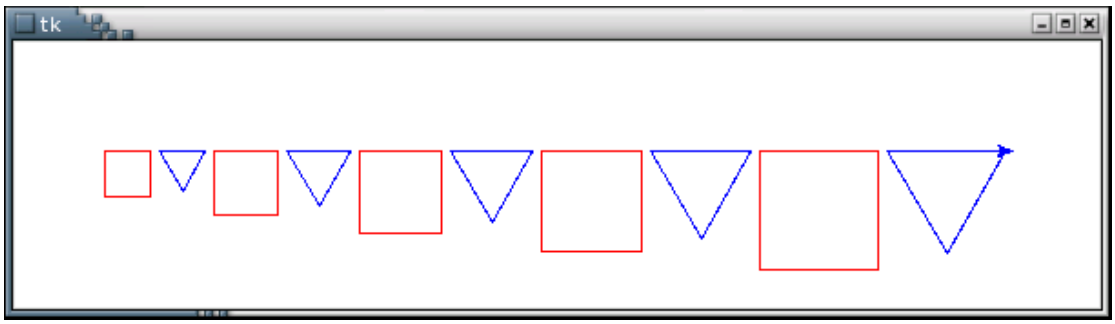
a) Que fait la fonction carre() ci-dessous ?

```
from turtle import *

def carre(taille,
couleur):
   color(couleur)
   c =0
   while c <4:
      forward(taille)
      right(90)
      c = c +1
```

b) Une fois que vous avez trouvé, vérifiez en écrivant ces lignes de code, (sauvegardez-les dans un fichier auquel vous donnerez le nom ***dessins_tortue.py***,) et exécutez votre script pour valider votre réponse en a)

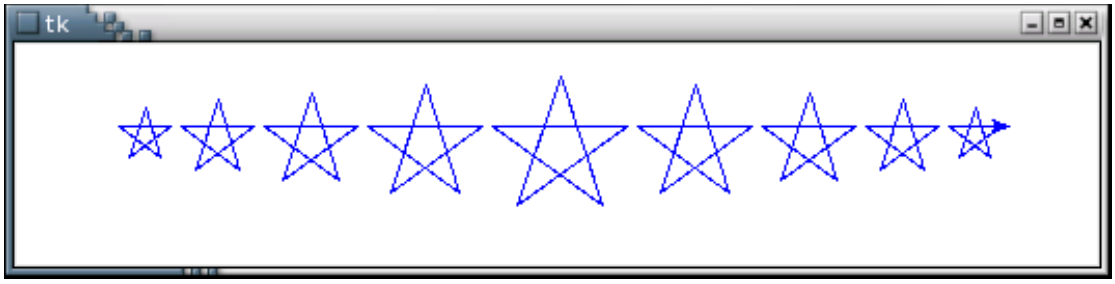
Exercice VII.2 : **Défi 1**

Complétez le module de fonctions graphiques dessins_tortue.py de l'exercice précédent.
Commencez par ajouter un paramètre angle à la fonction carre(), de manière à ce que les carrés puissent être tracés dans différentes orientations.
Définissez ensuite une fonction triangle(taille, couleur, angle) capable de dessiner un triangle équilatéral d'une taille, d'une couleur et d'une orientation bien déterminées.
Testez votre module à l'aide d'un programme qui fera appel à ces fonctions à plusieurs reprises, avec des arguments variés pour dessiner une série de carrés et de triangles :



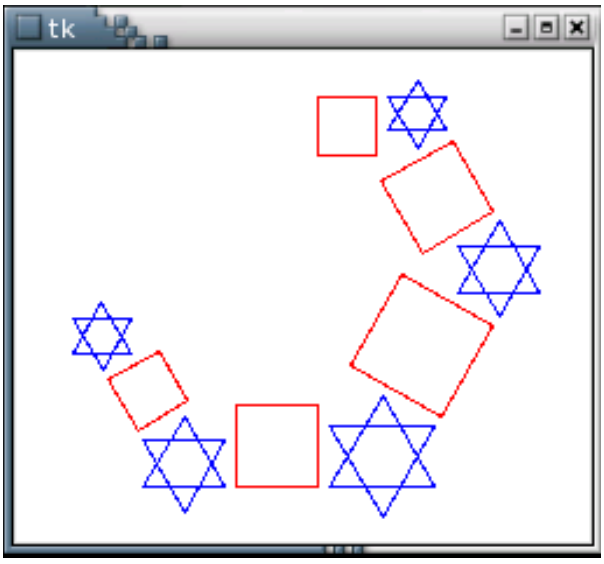
Exercice VII.3 : **Défi 2**

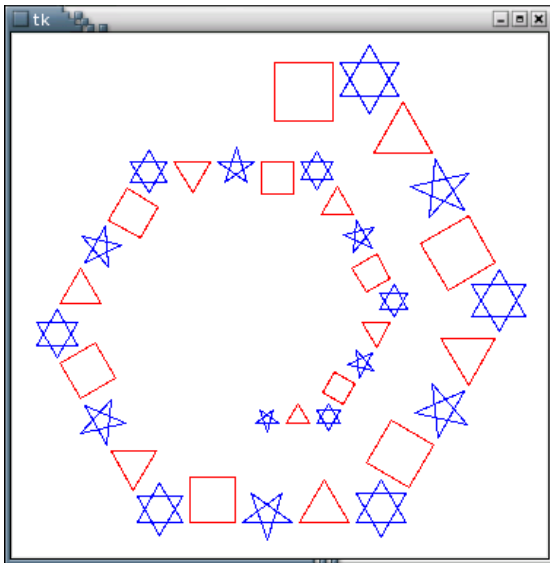
Ajoutez au module de l'exercice précédent une fonction **etoile5()** spécialisée dans le dessin d'étoiles à 5 branches. Dans votre programme principal, insérez une boucle qui dessine une rangée horizontale de de 9 petites étoiles de tailles variées :



Exercice VII.4 : **Défi 3**

Ajoutez au module de l'exercice précédent une fonction etoile6() capable de dessiner une étoile à 6 branches, elle-même constituée de deux triangles équilatéraux imbriqués. Cette nouvelle fonction devra faire appel à la fonction triangle() définie précédemment.
Votre programme principal dessinera également une série de ces étoiles :





REMARQUE VII.4.A : **Méthodes Turtle et Screen disponibles**

https://docs.python.org/release/3.1.3/library/turtle.html

Turtle motion

Move and draw

```
forward() | fd()
backward() | bk() | back()
right() | rt()
left() | lt()
goto() | setpos() | setposition()
setx()
sety()
setheading() | seth()
home()
circle()
dot()
stamp()
clearstamp()
clearstamps()
undo()
speed()
```

Tell Turtle's state

```
position() | pos()
towards()
xcor()
ycor()
heading()
distance()
```

Setting and measurement

```
degrees()
radians()
```

Pen control

Drawing state

```
pendown() | pd() | down()
penup() | pu() | up()
pensize() | width()
pen()
isdown()
```

Color control

```
color()
pencolor()
fillcolor()
```

Filling

```
filling()
begin_fill()
end_fill()
```

More drawing control

```
reset()
clear()
write()
```

Turtle state

Visibility

```
showturtle() | st()
hideturtle() | ht()
isvisible()
```

Appearance

```
shape()
resizemode()
shapesize() | turtlesize()
shearfactor()
settiltangle()
tiltangle()
tilt()
shapetransform()
get_shapepoly()
```

Using events

```
onclick()
onrelease()
ondrag()
```

Special Turtle methods

```
begin_poly()
end_poly()
get_poly()
clone()
getturtle() | getpen()
getscreen()
setundobuffer()
undobufferentries()
```

TurtleScreen

Window control

```
bgcolor()
bgpic()
clear() | clearscreen()
reset() | resetscreen()
screensize()
setworldcoordinates()
```

Animation control

```
delay()
tracer()
update()
```

Using screen events

```
listen()
onkey() | onkeyrelease()
onkeypress()
onclick() | onscreenclick()
ontimer()
mainloop()
```

Settings and special methods

```
mode()
colormode()
getcanvas()
getshapes()
register_shape() | addshape()
turtles()
window_height()
window_width()
```

Input methods

```
textinput()
numinput()
```

Methods specific to Screen

```
bye()
exitonclick()
setup()
title()
```

Série VIII : Les listes

But : Le but de cette série est de généraliser ce que nous avons vu avec des chaînes de caractères à d'autres types de chaînes de données : les listes.

Exercice VIII.0 : **Le type de données "list"***

Décrire les objets de type list en exécutant le script suivant :

```
Li1 = [ 1 , 2 , 3 ]
print (type ( Li1 ))
print (Li1)
#
Li2 =[27 , "Bonjour  comment  allez-vous  ?" , 3.1415 , Li1 ]
print (type ( Li2 ))
print (Li2)
#
Li3=range ( 1 , 4 , 1 )
print (type ( Li3 ))
print (Li3)
#
Li4 = [ ]
print (type ( Li4 ))
print (Li4)
#
Li5 =[ Li1 , Li2 , Li3 , Li4 ]
print (type ( Li5 ))
print (Li5)
```

Les éléments d'un objet de type "list" doivent-ils tous être du même type ?

Exercice VIII.1 : **L'instruction "list"***

Déterminer l'action de l'instruction list en exécutant le script suivant :

```
txt="Bonjour ,  comment  allez-vous  ?"
Li = list( txt )
print( Li )
print(type( txt ))
print(type( Li ))
```

Exercice VIII.2 : **Opérations sur les listes de nombres***

Déterminer l'action des fonctions `min`, max et sum sur les listes de nombres en exécutant le script suivant :

```
Li = [ -2, 3.14, 37, -786.2 ]
print(Li)
print(min( Li ))
print(max( Li ))
print(sum( Li ))
```

Exercice VIII.3 : **Opérations sur les listes***

Déterminer l'action de la fonction `len` et des opérations `+` et sur les listes en exécutant le script suivant :

```
Li1 = [ 1, 2, " bonjour " ]
Li2 = [ [ 3, list( "texte" ) ] , [ ] , 7.56 , "a" ]

print(Li1)
print(Li2)
print(len ( Li1 ) ,  len ( Li2 ))

Li = Li1 + Li2
print(Li)

Li3 = 2 * Li1
print(Li3)
```

Exercice VIII.4 : **Test d'appartenance***

Déterminer l'action des opérateurs `in` et not in sur une liste en exécutant le script suivant :

```
texte="Bonjour, comment allez-vous  ?"
Li = list(texte) + [[ 1, 2, 3 ], 7 ]
print(Li)

print("o"  in  Li)
print("o"  in texte)
print(1  in  Li)
print(7 not in  Li)
```

Exercice VIII.5 : **Éléments d'une liste***

Déterminer le résultat de l'instruction Li[i : j :k] pour une liste Li en exécutant le script suivant :

```
texte = "Bonjour, comment allez-vous ?"
Li = list( texte )
print("a :", Li)
print("b :", len(Li))

print("c :", Li[0])
print("d :", Li[5])

print("e :", Li[1])
print("f :", Li[3])

print("g :", Li[3 : 9 : 2 ])
print("h :", Li[3 : ])
print("i :", Li[: 3 ])

print("j :", Li[ : : 2 ])
print("k :", Li[ 1 :  1 : 2 ])
print("l :", Li[ : :  1 ])
```

REMARQUE VIII.5.A : **Chaînes, listes, tuples, dictionnaires, indice,…**

Sous Python, on peut définir divers types de données composites, connues sous le nom générique de **séquences,** utiles à regrouper de manière structurée des ensembles de valeurs : les listes, les tuples, les dictionnaires. Une liste est une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets. Comme les chaînes de caractères, les listes sont des séquences, c'est-à-dire des collections ordonnées d'objets. Les divers éléments qui constituent une liste sont en effet toujours disposés dans le même ordre, et l'on peut donc accéder à chacun d'entre eux individuellement si l'on connaît son index dans la liste (la numérotation de ces index commence à partir de zéro, et non à partir de un). À la différence de ce qui se passe pour les chaînes, qui constituent un type de données non-modifiables, il est possible de changer les éléments individuels d'une liste.

Exercice VIII.6 : **Modulo**

Analyser ce petit script, puis vérifier son fonctionnement :

```
jour = ['dimanche','lundi','mardi','mercredi','jeudi','vendredi','samedi']
a, b = 0, 0
while (a < 25):
     a = a + 1
     b = a % 7
     print('{:3d}'.format(a), jour[b])  # c.f. https://pyformat.info/
```

Exercice VIII.7 : **Opérations sur un élément d'une liste****

Modification :

```
>>> jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> jour[3] = jour[3] +47
>>> print(jour)
['lundi', 'mardi', 'mercredi', 1847, 20.357, 'jeudi', 'vendredi']
```

Substitution :

```
>>> jour[3] = 'Juillet'
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'Juillet', 20.357, 'jeudi', 'vendredi']
```

Suppression :

```
>>> del(jour[4])
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'juillet', 'jeudi', 'vendredi']
```

Ajout :

```
>>> jour.append('samedi')
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'juillet', 'jeudi', 'vendredi', 'samedi']
```

REMARQUE VIII.7.A : **Fonction ou méthode ?**

`del()` et `append()` ont des actions de même nature sur une liste (ajout ou suppression d'un élément) mais font appel à des concepts informatiques bien différents. `del()` est une fonction intégrée. Pour `append()`, il faut considérer que la liste est un objet, dont on va utiliser l'une des méthodes. Les concepts informatiques d'objet et de méthode font partie de la programmation orientée objet (POO) qui n'a pas été expliquée, mais cela n'empêche pas de comprendre « comment faire » et de l'utiliser dans les programmes.

Un peu de vocabulaire :

Nous avons appliqué la méthode `append()` à l'objet jour, avec l'argument 'samedi'.

La méthode `append()` est une fonction qui est attachée aux objets du type « liste ». Il existe plusieurs méthodes (c'est-à-dire des fonctions attachée, ou plutôt « encapsulées » dans les objets de type « liste »). Notons simplement au passage que l'on applique une méthode à un objet en reliant les deux à l'aide d'un point. (D'abord le nom de la variable qui référence l'objet, puis le point, puis le nom de la méthode, cette dernière toujours accompagnée d'une paire de parenthèses.)

La méthode `remove()` est aussi très intéressante.

Exercice VIII.8 : **Fusion organisée de listes**

```
t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
t2 = ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin',\
      'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre']
```

Écrivez un petit programme qui crée une nouvelle liste t3. Celle-ci devra contenir tous les éléments des deux listes en les alternant, de telle manière que chaque nom de mois soit suivi du nombre de jours correspondant : ['Janvier', 31, 'Février', 28, 'Mars', 31, etc.].

Exercice VIII.9 : **Afficher proprement**

Écrivez un programme qui affiche « proprement » tous les éléments d'une liste. Si on l'appliquait par exemple à la liste t2 de l'exercice ci-dessus, on devrait obtenir :
Janvier Février Mars Avril Mai Juin Juillet Août Septembre Octobre Novembre Décembre

Exercice VIII.10 : **Le plus grand vaut …**

Écrivez un programme qui recherche le plus grand élément présent dans une liste donnée. Par exemple, si on l'appliquait à la liste [32, 5, 12, 8, 3, 75, 2, 15], ce programme devrait afficher :
le plus grand élément de cette liste a la valeur 75.
Si c'est trop facile, rechercher les deux plus grands éléments de la liste.

Exercice VIII.11 : **Séparer en diverses listes**

Écrivez un programme qui analyse un par un tous les éléments d'une liste de nombres (par exemple celle de l'exercice précédent) pour générer deux nouvelles listes. L'une contiendra seulement les nombres pairs de la liste initiale et l'autre, les nombres impairs. Par exemple, si la liste initiale est celle de l'exercice précédent, le programme devra construire une liste : "pairs" qui contiendra [32, 12, 8, 2], et une liste : "impairs" qui contiendra [5, 3, 75, 15]. Astuce : pensez à utiliser l'opérateur modulo (%).

Exercice VIII.12 : **Séparer en diverses listes (bis)**

Écrivez un programme qui analyse un par un tous les éléments d'une liste de mots (par exemple : ['Jean', 'Maximilien', 'Brigitte', 'Sonia', 'Jean-Pierre', 'Sandra']) pour générer deux nouvelles listes. L'une contiendra les mots comportant moins de 6 caractères, l'autre les mots comportant 6 caractères ou davantage.

Exercice VIII.13 : **Le plus grand est placé …**

Définissez une fonction `indexMax(liste)` qui renvoie l'index de l'élément ayant la valeur la plus élevée dans la liste transmise en argument. Exemple d'utilisation :

```
serie = [5, 8, 2, 1, 9, 3, 6, 7]
print(indexMax(serie))
4
```

Corrigé Les listes

Solution Exercice VIII.0 :

Les objets de type list sont des listes. Les éléments d'une liste sont des objets quelconques et peuvent être de types différents.

Solution Exercice VIII.1 :

L'instruction list ("texte") crée la liste ['t','e','x','t','e'].

Solution Exercice VIII.2 :

La fonction min(l) renvoie le plus petit nombre dans la liste l, max(l) donne le nombre le plus grand et sum(l) calcule la somme des nombres contenus dans la liste l.

Solution Exercice VIII.3 :

La fonction len(l1) donne le nombre d'éléments contenus dans la liste l1. L'opération l1 +l2 donne la concaténation de l1 et l2 et 2∗l1 donne la concaténation de la liste l1 avec elle-même.

Solution Exercice VIII.4 :

L'opérateur x in l teste si l'objet x est égal à l'un des éléments de la liste l. Il renvoie True en cas de succès et False sinon. L'opérateur x not in l est équivalent à not (x in l).

Solution Exercice VIII.5 :

Nous constatons que
1. Li[0] donne le premier élément de Li,
2. Li[5] donne le sixième élément de Li,
3. Li[−1] donne le dernier élément de Li,
4. Li[−3] donne le 27e élément de Li ,
5. Li [3:9:2] donne une tranche d'éléments de l partant du troisième élément jusqu'au huitième avec un pas de 2,
6. Li[3:] donne tous les éléments de Li à partir du troisième,
7. Li[:−3] donne tous les éléments de Li jusqu'au 27e,
8. Li[::2] donne tous les éléments de Li ayant un indice pair,
9. Li[1:−1:2] donne tous les éléments de Li ayant un indice impair,
10. Li[::−1] donne tous les éléments de Li, mais dans l'ordre inverse.

Série IX : La bonne structure d'un programme

```
#!/usr/bin/python3
# -*-coding:utf-8 -*

#=====================================================================#
#---------------------------------------------------------------------#
# NOM DU PROJET                                                       #
#---------------------------------------------------------------------#
#*********************************************************************#
# Auteur(s) - Date de creation                                        #
# licence(s)                                                          #
#---------------------------------------------------------------------#
# Notes/Commentaires                                                  #
#                                                                     #
#---------------------------------------------------------------------#
# HISTORIQUE (le plus recent en premier)                              #
#                                                                     #
#                                                                     #
# n° version                                                          #
# Motif de la modification et/ou commentaire bref                     #
#                                                                     #
#                                                                     #
#=====================================================================#

#-------------------------------------------#
# Importation des packages                  #
#-------------------------------------------#
import time


#-------------------------------------------#
# Declaration des variables                 #
#-------------------------------------------#
chaine = "**"

#-------------------------------------------#
# Procedures et fonctions                   #
#-------------------------------------------#
def ma_fonction(chaine):
    """affiche chaine sur le terminal"""
    print(a,end="")

#-------------------------------------------#
# Boucle principale                         #
#-------------------------------------------#
for i in range (0,15): # affiche "chaine" toutes les 1 seconde
    ma_fonction(chaine)
    time.sleep(1)
print()
```

REMARQUE IX.0.A : **Le statut spécial des 2 premières lignes d'un fichier**

En premier lieu, inclure les **pseudo-commentaires** suivants au début de tous vos scripts (obligatoirement à la 1ère ou à la 2e ligne), le premier indique l'emplacement de l'interpréteur PYTHON, version 3, le second le type d'encodage des caractères à utiliser :

```
#!/usr/bin/python3
# -*-coding:utf-8 -*
```

Passé ces deux premières lignes, la suite de la mise en page n'est pas utile à l'exécution du code donc souvent ignorée… jusqu'à ce qu'on s'en morde les doigts en réalisant qu'elle est tellement importante pour la maintenance du code !
Alors, on tiendra (à la main en ce qui nous concerne ou, dans le monde professionnel, grâce à un logiciel de gestion des révisions) à jour un cartouche complet en tête du fichier, ainsi que des minis cartouches résumant l'utilité/l'action du code pour chaque section.

Pour l'importation, il est possible d'importer des fonctions personnelles à l'aide de "import fichier" où fichier est un "fichier.py" se trouvant dans le répertoire de travail.

Exercices : Bibliothèque de fonctions.

Nous allons créer plusieurs fonctions (voir ci-après) dans un même fichier que nous nommerons fonctions.py. Prendre soin de bien structurer et commenter.

Exercice IX.1 : **Calcul du volume d'une boîte***

Créez la fonction `volBoite(x1,x2,x3)`, de manière à ce qu'elle puisse être appelée avec trois, deux, un seul, ou même aucun argument. Utilisez pour ceux-ci des valeurs par défaut égales à 10, à préciser lors de la définition de la fonction : `def volBoite(x1=10, x2=10, x3=10):`

Par exemple :

```
print(volBoite())          # doit donner le résultat : 1000
print(volBoite(5.2))       # doit donner le résultat : 520.0
print(volBoite(5.2, 3))    # doit donner le résultat : 156.0
```

Exercice IX.2 : **Autre calcul du volume d'une boîte***

Créez la fonction `volBoiteAutre(x1,x2,x3)`, ci-dessus de manière à ce qu'elle puisse être appelée avec un, deux, ou trois arguments. Si un seul est utilisé, la boîte est considérée comme cubique (l'argument étant l'arête de ce cube). Si deux sont utilisés, la boîte est considérée comme un prisme à base carrée (auquel cas le premier argument est le côté du carré, et le second la hauteur du prisme). Si trois arguments sont utilisés, la boîte est considérée comme un parallélépipède. Par exemple :

```
print(volBoiteAutre())             # doit donner le résultat : -1 (indication d'une erreur)
print(volBoiteAutre(5.2))          # doit donner le résultat : 140.608
print(volBoiteAutre(5.2, 3))       # doit donner le résultat : 81.12
print(volBoiteAutre(5.2, 3, 7.4))  # doit donner le résultat : 115.44
```

Exercice IX.3 : **Changer des caractères***

Définissez une fonction `changeCar(ch, ca1, ca2, debut, fin)` qui remplace tous les caractères ca1 par des caractères ca2 dans la chaîne de caractères ch, à partir de l'indice debut et jusqu'à l'indice fin, ces deux derniers arguments pouvant être omis (et dans ce cas la chaîne est traitée d'une extrémité à l'autre). Exemples de la fonctionnalité attendue :

```
>>> phrase = 'Ceci est une toute petite phrase.'
>>> print(changeCar(phrase, ' ', '*'))
Ceci*est*une*toute*petite*phrase.
>>> print(changeCar(phrase, ' ', '*', 8, 12))
Ceci est*une*toute petite phrase.
>>> print(changeCar(phrase, ' ', '*', 12))
Ceci est une*toute*petite*phrase.
>>> print(changeCar(phrase, ' ', '*', fin = 12))
Ceci*est*une*toute petite phrase.
```

Exercice IX.4 : **Programme principal***

Dans un nouveau fichier, appelé `principal.py`, demander à l'utilisateur s'il veut changer des caractères dans une phrase ou s'il veut calculer (et comment) le volume d'une boîte.
Appeler alors les fonctions créées précédemment.
Prendre soin de bien structurer et commenter.

Série X : Lecture et écriture dans des fichiers

But : Ouvrir, lire et écrire dans des fichiers de l'ordinateur, afin de sauvegarder des données pour les utiliser d'une session à l'autre de notre programme.

!! ATTENTION !!

> Nous allons travailler sur des répertoires et des fichiers, autrement dit sur le disque dur de l'ordinateur et non pas en mémoire vive. Les données seront créées, modifiées ou effacées de façon définitive. **Un retour en arrière n'est pas possible**, même en redémarrant le programme ou l'ordinateur.

Exercice X.0 : **Une première écriture dans un fichier**

Sous Python, l'accès aux fichiers est assuré par l'intermédiaire d'un objet-interface particulier, que l'on appelle *objet-fichier*. On crée cet objet à l'aide de la fonction intégrée **open()**. Celle-ci renvoie un objet doté de méthodes spécifiques, qui vous permettront de lire et écrire dans le fichier.
L'exemple ci-après vous montre comment ouvrir un fichier en écriture, y enregistrer deux chaînes de caractères, puis le refermer. Notez bien que si le fichier n'existe pas encore, il sera créé automatiquement. Par contre, si le nom utilisé concerne un fichier préexistant qui contient déjà des données, les caractères que vous y enregistrerez viendront s'ajouter à la suite de ceux qui s'y trouvent déjà. Vous pouvez faire tout cet exercice directement à la ligne de commande :

Sélectionnez

```
>>> obFichier = open('Monfichier','a')
>>> obFichier.write('Bonjour, fichier !')
>>> obFichier.write("Quel beau temps, aujourd'hui !")
>>> obFichier.close()
>>>
```

- La première ligne crée l'*objet-fichier***obFichier**, lequel fait référence à un fichier véritable (sur disque ou disquette) dont le nom sera **Monfichier**. Attention : *ne confondez pas le nom de fichier avec le nom de l'objet-fichier* qui y donne accès ! À la suite de cet exercice, vous pouvez vérifier qu'il s'est bien créé sur votre système (dans le répertoire courant) un fichier dont le nom est **Monfichier** (et dont vous pouvez visualiser le contenu à l'aide d'un éditeur quelconque).

- La fonction **open()** attend deux arguments, qui doivent tous deux être des chaînes de caractères. Le premier argument est le nom du fichier à ouvrir, et le second est le mode d'ouverture. **'a'** indique qu'il faut ouvrir ce fichier en mode « ajout » (*append*), ce qui signifie que les données à enregistrer doivent être ajoutées à la fin du fichier, à la suite de celles qui s'y trouvent éventuellement déjà. Nous aurions pu utiliser aussi le mode **'w'** (pour *write*), mais lorsqu'on utilise ce mode, Python crée toujours un nouveau fichier (vide), et l'écriture des données commence à partir du début de ce nouveau fichier. S'il existe déjà un fichier de même nom, celui-ci est effacé au préalable.

- La méthode **write()** réalise l'écriture proprement dite. Les données à écrire doivent être fournies en argument. Ces données sont enregistrées dans le fichier les unes à la suite des autres (c'est la raison pour laquelle on parle de fichier à accès séquentiel). Chaque nouvel appel de **write()** continue l'écriture à la suite de ce qui est déjà enregistré.

- La méthode **close()** referme le fichier. Celui-ci est désormais disponible pour tout usage.

Exercice X.1 : **Et une première lecture d'un fichier**

Vous allez maintenant rouvrir le fichier, mais cette fois en lecture, de manière à pouvoir y relire les informations que vous avez enregistrées dans l'étape précédente :

Sélectionnez
```
>>> ofi = open('Monfichier.txt', 'r')
>>> leTexte = ofi.read()
>>> print(leTexte)
Bonjour, fichier ! Quel beau temps, aujourd'hui !
>>> ofi.close()
```

Comme on pouvait s'y attendre, la méthode **read()** lit les données présentes dans le fichier et les transfère dans une variable de type chaîne de caractères (*string*). Si on utilise cette méthode sans argument, la totalité du fichier est transférée.
Le fichier que nous voulons lire s'appelle **Monfichier.txt**. L'instruction d'ouverture de fichier devra donc nécessairement faire référence à ce nom-là. Si le fichier n'existe pas, nous obtenons un message d'erreur. Exemple :

```
>>> ofi = open('Monficier.txt','r')
IOError: [Errno 2] No such file or directory: 'Monficier.txt'
```

- Par contre, nous ne sommes tenus à aucune obligation concernant le nom à choisir pour *l'objet-fichier*. C'est un nom de variable quelconque. Ainsi donc, dans notre première instruction, nous avons choisi de créer un objet-fichier **ofi**, faisant référence au fichier réel **Monfichier.txt**, lequel est ouvert en lecture (argument **'r'**).

- Les deux chaînes de caractères que nous avions entrées dans le fichier sont à présent accolées en une seule. C'est normal, puisque nous n'avons fourni aucun caractère de séparation lorsque nous les avons enregistrées. Nous verrons un peu plus loin comment enregistrer des lignes de texte distinctes.

- La méthode **read()** peut également être utilisée avec un argument. Celui-ci indiquera combien de caractères doivent être lus, à partir de la position déjà atteinte dans le fichier :

```
>>> ofi = open('Monfichier.txt', 'r')
>>> leTexte = ofi.read(7)
>>> print(leTexte)
Bonjour
>>> leTexte = ofi.read(15)
>>> print(t)
, fichier !Quel
```

  S'il ne reste pas assez de caractères au fichier pour satisfaire la demande, la lecture s'arrête tout simplement à la fin du fichier :

```
>>> leTexte = ofi.read(1000)
>>> print(leTexte)
beau temps, aujourd'hui !
```

Si la fin du fichier est déjà atteinte, **read()** renvoie une chaîne vide :

```
>>> leTexte= ofi.read()
>>> print(t)
```

- N'oubliez pas de refermer le fichier après usage :

```
>>> ofi.close()
```

Pour aller plus loin…

REMARQUE X.1.A : **L'arborescence**

Dans tout ce qui précède, nous avons admis sans explication que Python savait où trouver les fichiers. En réalité, il faut le préciser à Python.
Nous allons voir ci-dessous comment faire…

Avant d'essayer des instructions, il faut s'assurer d'avoir compris l'arborescence utilisée par le système (operating system).

Exercice X.2 : **Déterminer le répertoire de travail courant**

Tout d'abord, il faut définir le répertoire de travail courant. Enfin celui que l'on veut vraiment utiliser, pas celui que Python aura choisi pour nous et dans lequel il n'aurait pas fallu faire de modifications[1]…

Pour cela depuis Windows créer un nouveau dossier, le nommer (p. exemple fichiersPourPython). Maintenant il est possible depuis le Shell de définir ce répertoire comme dossier de travail courant en utilisant chdir (change directory), du module os :

Soit à l'aide d'un chemin absolu (on donne l'arborescence depuis la lettre du répertoire p. ex C:/) :

```
>>> import os
>>> os.chdir("C:/fichiersPourPython")
>>>
```

Soit à l'aide d'un chemin relatif (on donne l'arborescence du nouveau répertoire depuis l'actuel):

```
>>> import os
>>> os.chdir("../fichiersPourPython")
>>>
```

Dans l'exemple ci-dessus on utilise ../ ce qui permet de « remonter » l'arborescence ../ désigne le répertoire parent.

1 Python choisit le répertoire courant de la façon suivante : pour le Shell, c'est le dossier où se trouve l'interpréteur, pour un script, c'est le dossier où se trouve le script, l'instruction os.getcwd() permet d'obtenir le répertoire courant.

Pour les essais qui viennent :

1. Depuis Windows, créez un sous-répertoire « **testsRWpython** »
2. Depuis le Shell, définissez-le comme répertoire de travail à l'aide de os.chdir ;
3. Depuis Windows (avec l'application bloc-notes, sans mise en forme), créez **fichierPourTests.txt** avec comme contenu « *Fais de ta vie un rêve et de ton rêve une réalité » (Antoine de Saint Exupéry)* »

REMARQUE X.2.A : **L'encodage**

Dans tout ce qui précède nous avons aussi admis sans explication que les chaînes de caractères étaient échangées telles quelles entre l'interpréteur Python et le fichier. En réalité, ceci est inexact, parce que les séquences de caractères doivent être converties en séquences d'octets pour pouvoir être mémorisées dans les fichiers, et il existe différentes normes pour cela. En toute rigueur, il faut donc préciser à Python la norme d'encodage utilisée dans vos fichiers.
Nous allons voir ci-dessous comment faire…

Exercice X.3 : **Ouverture du fichier**

Le script doit tout d'abord demander au système l'accès au fichier, la built-in function de Python est open, qui renvoie un objet doté de méthodes spécifiques, qui permettent de lire et écrire dans le fichier : open(file, mode='r', encoding='UTF-16')

file = le nom du fichier (avec, s'il n'est pas dans le répertoire de travail, son chemin absolu ou relatif)

mode = 'r' (lecture) , 'w' (détruire, (re)créer, puis écriture), 'a' (si nécessaire créer, puis ajout à la fin du fichier) ou 'rb', 'wb', 'ab' pour un fichier binaire.

encoding= 'UTF-16' ou 'Latin-1' ou 'US-ASCII' ou bien d'autres normes trouver la bonne…

Définissons « fichierL », regardons le contenu de cette variable et son type :

```
>>> fichierL=open('fichierPourTests.txt', 'r', encoding='UTF-16')
>>> fichierL
<_io.TextIOWrapper name='fichierPourTests.txt' mode='r' encoding='UTF-16'>
>>> type(fichierL)
<class '_io.TextIOWrapper'>
```

Exercice X.4 : **Lire l'intégralité du fichier**

Pour ce faire, on utilise la méthode read de la classe TextIoWrapper. Elle renvoie l'intégralité du fichier –ici dans la variable contenu- :

```
>>> contenu=fichierL.read()
>>> contenu
'« Fais de ta vie un rêve et de ton rêve une réalité »\n(Antoine de Saint Exupéry)\n'
```

Exercice X.5 : **Fermer le fichier**

Finalement, on doit toujours fermer le fichier pour indiquer au système que d'autres applications peuvent y accéder, la méthode close :

```
>>> fichierL.close()
```

Exercice X.6 : **Écriture dans un fichier**

Le principe est simple, après avoir ouvert un fichier, on peut utiliser le mode w ou le mode a. Le premier écrase le contenu éventuel du fichier, alors que le second ajoute ce que l'on écrit à la fin du fichier. Ces deux modes créent le fichier s'il n'existe pas. La méthode write retourne le nombre de caractères écrits. Enfin, ils ne le seront réellement que lorsque le fichier sera fermé ou passé en mode lecture.

```
>>> fichierL=open('nouveauFichierPourTests.txt', 'w',encoding='UTF-16')
>>> fichierL
<_io.TextIOWrapper name='nouveauFichierPourTests.txt' mode='w' encoding='UTF-16'>
>>> fichierL.write("Petit test d'écriture. Eh éh éh !")
33
>>> fichierL.close()
>>> fichierL=open('nouveauFichierPourTests.txt', 'r',encoding='UTF-16')
>>> cont=fichierL.read()
>>> cont
"Petit test d'écriture. Eh éh éh !"
```

Et pour ajouter de nouvelles informations à ce même fichier :

```
>>> fichierL=open('nouveauFichierPourTests.txt', 'a',encoding='UTF-16')
>>> fichierL.write("\nEt voilà une nouvelle ligne!")
29
>>> fichierL=open('nouveauFichierPourTests.txt', 'r',encoding='UTF-16')
>>> cont=fichierL.read()
>>> cont
"Petit test d'écriture. Eh éh éh !\nEt voilà une nouvelle ligne!"
>>>
```

Print() permet les sauts de lignes etc :

```
>>> print(cont)
Petit test d'écriture. Eh éh éh !
Et voilà une nouvelle ligne!
```

Ne pas oublier de fermer le fichier

```
>>> fichierL.close()
```

REMARQUE X.6.A : **Écrire d'autres types de données**

La méthode write n'accepte en paramètre que des chaînes de caractères. Si vous voulez écrire dans votre fichier des nombres, des scores par exemple, il vous faudra les convertir en chaîne avant de les écrire et les convertir en entier après les avoir lus.

Et pour les curieux qui souhaitent gérer des fichiers ou répertoires (création, suppression, …), un coup d'œil sur une documentation du module os devrait les intéresser…

À venir peut-être :

Série XI : Traitement d'image

Série XII : Codage de l'information

Série XIII : Algorithmique