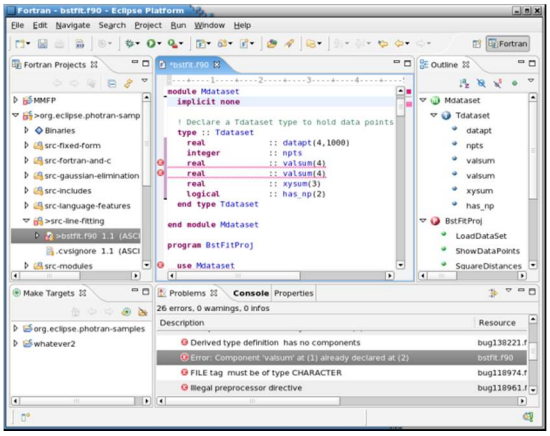
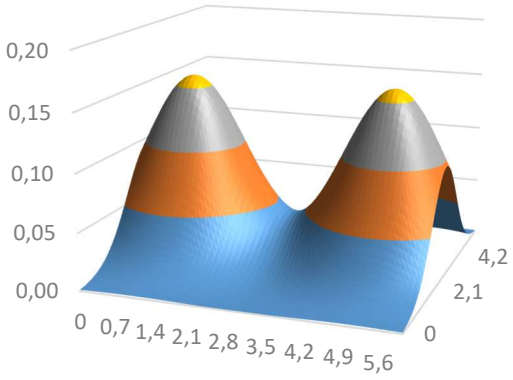
Atelier de Recherche Encadrée :
livret de l'étudiant

Introduction à la **T**opologie en **S**ciences (**ITS**)





La topologie est une branche des mathématiques portant sur des ensembles munis d'une notion de voisinage autour de chaque point, et qu'on appelle domaines topologiques. La théorie des systèmes dynamiques gradient est une technique qui nous permet de regrouper ces points en plusieurs domaines qui possèdent les mêmes caractéristiques. La répartition tridimensionnelle des points peut être élaborée sur une grille régulière et les valeurs de ces points par la loi normale standard ou une approche stochastique. Un programme informatique simple est alors possible afin de déterminer les valeurs de la fonction ainsi que les domaines topologiques.

Les informations concernant les A.R.E sont affichées sur les panneaux au sous-sol, niveau Saint Bernard près de la cafétéria, niveau SB dans couloir jaune.

Barème de l'atelier (100pts)

- Note de Contrôle Continu est composée de :

  (Auto-)Apprentissage de connaissance /20

  En mathématiques et en informatique, l'étudiant devra répondre à des QCM (3*7pts) portant sur les notions abordées la semaine précédente.

  Restitution et défense / 30

  Le projet ITS sera présenté, à l'oral, par les étudiants.

- Note de TP est composée de :

  Suivi du travail personnel /10

  La ponctualité, l'investissement et le comportement en classe feront l'objet d'une évaluation.

  Démarche et investigation scientifique / 40

  Le projet informatique sera évalué par rapport aux critères suivants : problématique, méthode pour sa résolution, typage des variables, analyse et optimisation de l'algorithme. Un rapport écrit sera demandé afin de répondre à tous ces critères, mais aussi pour expliquer le code informatique créé et démontrer sa validité scientifique. Une bibliographie sera demandée.

L'ARE ITS c'est trois conférences en mathématiques, trois ateliers sur machines (TME) en informatique, une séance de recherche bibliographique avec un membre de la bibliothèque de Licence 1. Ces activités, en présentiel, représentent 15h de cette UE à 3ECTS (donc un total de 30h). Les heures restantes sont réservées pour le travail personnel en non présentiel. Ce travail est effectué en groupe pour le projet et la soutenance.

En mathématiques, les conférences seront un support bénéfique pour le projet final. À l'aide d'exemples et de codes Python, le conférencier présentera les notions suivantes : les systèmes dynamiques gradient, la loi normale standard et les approches stochastiques. Ces outils mathématiques seront utilisés pour le rapport de recherche bibliographique et pour le projet informatique.

En informatique, les séances seront à la fois une remise à niveau des connaissances de programmation (basées sur les connaissances acquises en LU1IN001) et aussi un moment

privilégié pour la réalisation du projet. Tout d'abord, un rappel de cours sera présenté et des exercices de programmation seront effectués.

Chaque semaine, un QCM sera effectué afin de valider vos connaissances (20pts). Les questions porteront sur les notions vues en mathématiques et en informatique la semaine précédente.

Une séance sera consacrée à la recherche bibliographique et à la validation d'un code informatique (afin d'éviter le plagia par exemple). Pour cela, un intervenant de la bibliothèque de Licence 1 présentera les outils à utiliser au cours de cet atelier.

Le projet ITS sera effectué en groupe. Le travail en groupe est important et les enseignants seront vigilants sur la part de travail effectué par chaque étudiant. Pour une programmation en groupe, les étudiants sont fortement incités à se répartir le travail de programmation grâce à l'utilisation des fonctions. Un code informatique est demandé ainsi que son dossier qui doit comporter une introduction, une description argumentée de la problématique, le code commenté, des explications sur son utilisation, une validation scientifique des résultats obtenus, une conclusion et une bibliographie (40pts).

L'atelier se terminera par une présentation orale (30pts). Le jury posera des questions sur ce qui aura été présenté.

Attention, la présence à tous les ateliers et à toutes les activités est obligatoire.

- Toute absence à une séance obligatoire devra être justifié auprès des enseignants de votre atelier ARE et ce dès la séance suivante (pas au secrétariat et n'attendez pas la fin du semestre).
- 2 absences non justifiées entrainent la non-validation de l'UE (note 0/100 aux deux sessions), sauf décision contraire du jury de l'UE et du parcours.
- Toute absence, justifiée ou non justifiée, lors d'un contrôle écrit ou oral entrainera une note de 0 à ce contrôle.
- Toute absence ou retard à la restitution d'un devoir ou d'un rapport entrainera une note de 0 à celui-ci.
- En cas de retard (réveil qui ne sonne pas ou train en retard), l'étudiant doit se présenter à la séance prévue même en retard. Il sera accepté ou non selon les cas (la décision revient à l'enseignant).
- La session de rattrapage n'est pas autorisée en cas de 2 ou plus d'absences non justifiées.

Planning 2020 ITS

Jour de l'ARE : mardi

Début des cours le 21/01/2020

Heure de début : 13h45

	ITS G1 : MIPI-21.1A		**ITS G2 : MIPI-21.1B**
21/01/2020	Introduction de l'atelier *atrium 257*		
28/01/2020	**Informatique 1** *atrium 100*	2h	**Mathématiques 1** *atrium 257*
04/02/2020	**Mathématiques 1** *atrium 257*	2h	**Informatique 1** *atrium 100*
11/02/2020	**Recherche bibliographique** *atrium 100*	2h	**Mathématiques 2** *atrium 257*
18/02/2020	**Informatique 2** *atrium RC15*	2h	**Recherche bibliographique** *atrium 121*
25/02/2020	**Mathématiques 2** *atrium 257*	2h	**Informatique 2** *atrium 435*
03/03/2020			
10/03/2020	**Informatique 3** *atrium 121*	2h	**Mathématiques 3** *atrium 257*
17/03/2020	**Mathématiques 3** *atrium 257*	1h45	**Informatique** 3 *atrium 232*
24/03/2020			
31/03/2020			
21/04/2020	**Rendu final** *atrium 257*	0h15	**Rendu final** *atrium 257*
28/04/2020	**Rendu final** *atrium 257*		**Rendu final** *atrium 257*

Mathématiques. Conférences.

1- Systèmes dynamique gradient

En science, un système dynamique est la donnée d'un système et d'une loi décrivant l'évolution de ce système. Ce peut être l'évolution d'une réaction chimique au cours du temps ou le mouvement des planètes dans le système solaire (régi par la loi universelle de la gravitation de Newton). L'espace des phases est une structure correspondant à l'ensemble de tous les états possibles du système considéré. Ce peut être un espace vectoriel, une variété différentielle, un espace mesurable… Un portrait de phase est une représentation géométrique des trajectoires d'un système dynamique dans l'espace des phases : à chaque ensemble de conditions initiales correspond une courbe ou un point.

2- Loi normale standard

En théorie des probabilités et en statistique, la loi normale est l'une des lois de probabilité les plus adaptées pour modéliser des phénomènes naturels issus de plusieurs événements aléatoires. Elle est en lien avec de nombreux objets mathématiques dont le mouvement brownien, le bruit blanc gaussien ou d'autres lois de probabilité. Elle est également appelée loi gaussienne, loi de Gauss ou loi de Laplace-Gauss des noms de Laplace (1749-1827) et Gauss (1777-1855), deux mathématiciens, astronomes et physiciens qui l'ont étudiée.

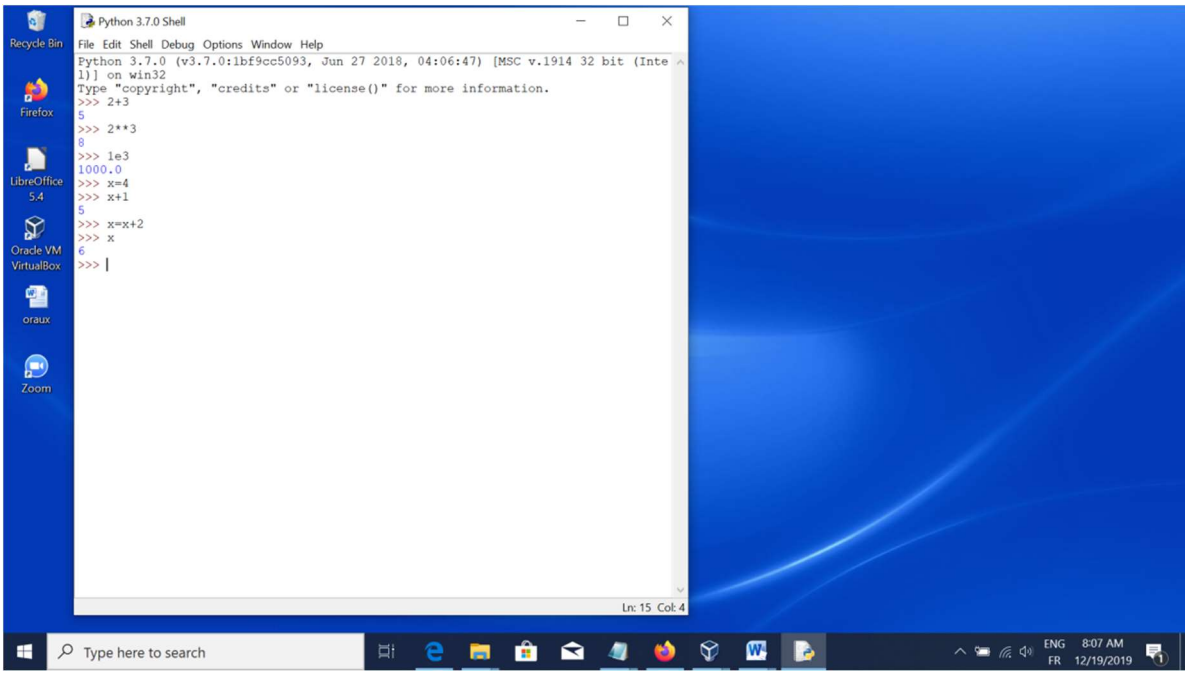
3- Approches stochastiques

Générer des nombres aléatoires pose deux problèmes : en premier lieu la génération elle-même et le second savoir caractériser le hasard. Dans un livre Cournot définit le hasard comme « *les événements amenés par la combinaison ou la rencontre de phénomènes qui appartiennent à des séries indépendantes, dans l'ordre de la causalité, sont ce qu'on nomme des événements fortuits ou des résultats du hasard* ». Cette définition est utilisable pour construire de bons générateurs de nombres aléatoires, par le croisement d'un générateur de nombres (ayant de « bonnes » propriétés, notamment la possibilité de générer tous les nombres parmi lesquels on veut faire le tirage) avec un générateur d'arrêt indépendant.

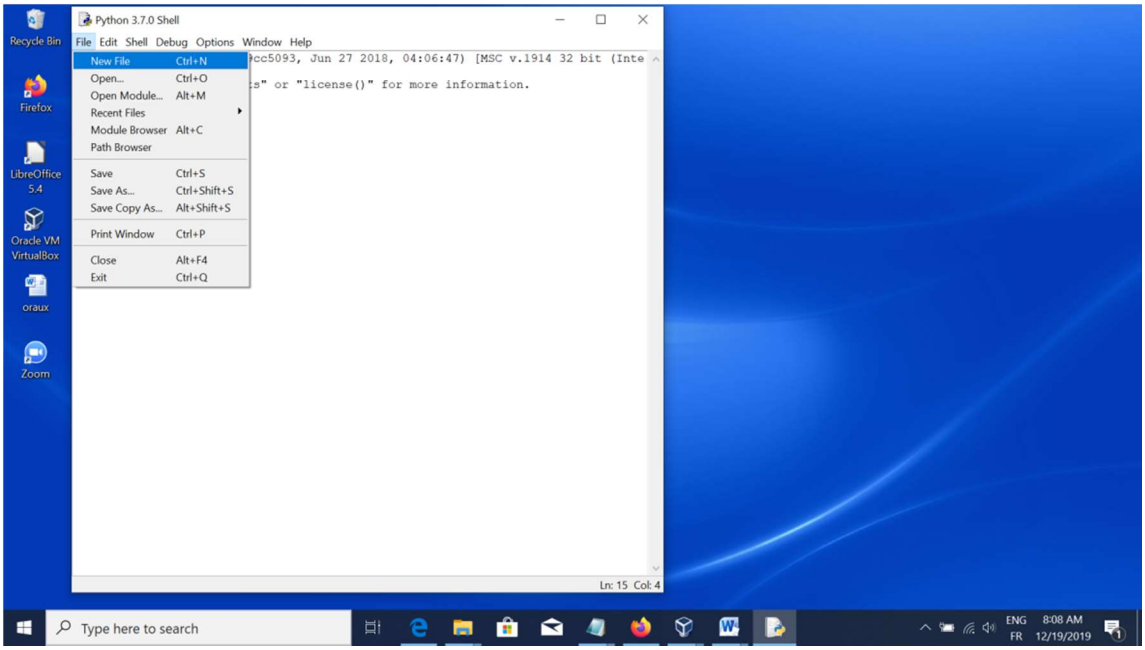
Python. Cours et exercices.

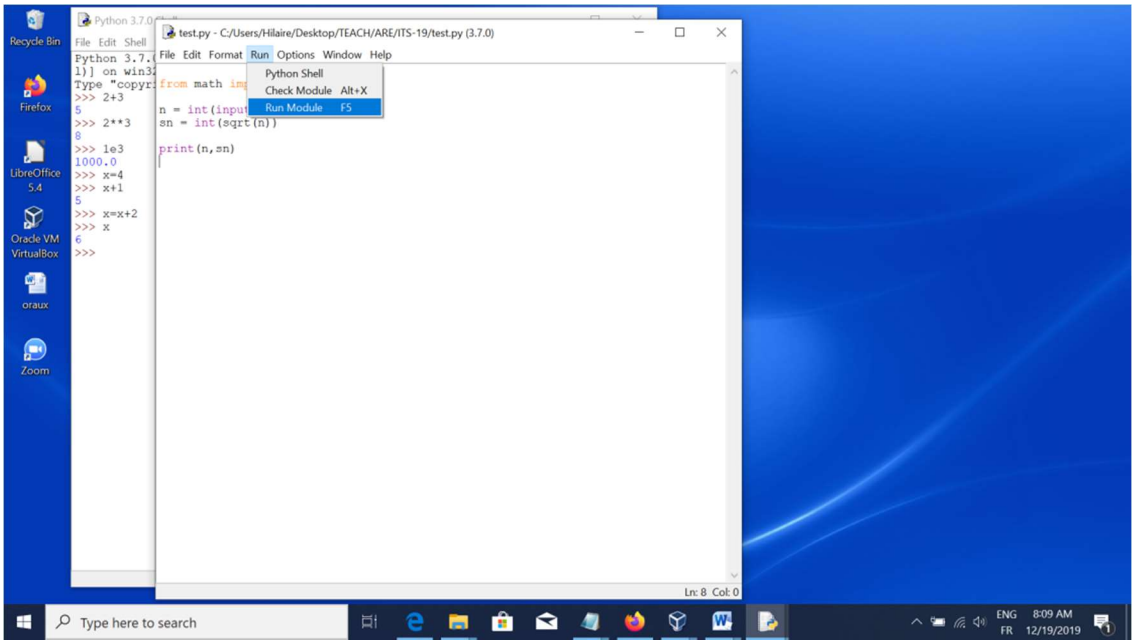
Cette présentation est une introduction au langage python. Nous allons utiliser l'interface de développement IDLE PYTHON 3.6 sur le bureau de l'utes (à trouver dans l'onglet outils). Vous pouvez vous connecter à distance sur le bureau de l'utes avec vos identifiants (https://lutes.upmc.fr/bdl-ext.php) ou bien vous pouvez installer sur vos ordinateurs le python qui contient idle à partir de la page https://www.python.org/downloads/.

Une fois l'interface IDLE lancée, vous avez une fenêtre interactive où vous pouvez taper des instructions qui seront calculées directement à chaque fois que vous tapez sur « entrée ».



Par la suite, il est préférable de réunir vos instructions dans un fichier (file/new ou file/open), ce qui permet de les sauvegarder et de les réutiliser plus tard. A l'avenir, un fichier correspondra à un programme. Après avoir sauvegardé votre fichier programme dans un fichier (file/save as ou file/save) vous pouvez l'exécuter à l'aide de run/run module, et les résultats s'affichent dans la fenêtre d'interface.





I. Les variables

Dans python, vous pouvez déclarer des variables. Par exemple i = 4, x=3.2, p = True… Et à la différence de beaucoup d'autres langages informatiques, le typage est automatique (entier, réel, booléen, chaîne de caractères). Le nom d'une variable ne doit pas contenir d'espace, d'accent, de symbole (sauf éventuellement underscore « _ ») et ne doit pas commencer par un chiffre. Dans la mesure du possible prenez des noms de variables explicites par exemple : pi = 3.14.

Le symbole « = » permet d'affecter une valeur à une variable, cette dernière correspondant à un emplacement mémoire. D'une manière générale, on utilise toujours des variables pour stocker des résultats intermédiaires de calcul.

II. Les opérations de base

Parmi les opérations possibles sur les nombres et sur les variables vous avez l'addition +, la soustraction -, la multiplication *, la division réelle /, la division entière // (3//4 donne 0, et 5//4 donne 1…), ** la puissance : 2**3 donne 8, et % le modulo, tel que a%b fournit le reste de la division entière de a par b (2%3 donne 2, car 0*3+2. 5%2 donne 1 car 2*2+1, 14%3 donne 2 car 4*3+2…)

A l'aide de l'instruction :

```
from math import *
```

Vous pouvez appeler les fonctions mathématiques : sqrt(x) pour la racine carrée, abs(x), pour la valeur absolue, exp, cos, sin, etc…

III. Les entrées-sorties

Afin d'interagir avec l'utilisateur, vous avez besoin de pouvoir écrire (print) et lire (input) dans la fenêtre d'interface. Un petit exemple ?...

```
nom = input(''entrer votre nom ? '')
print(''bonjour '', nom)
```

Mais attention, la fonction input lit à l'écran une valeur, et retourne dans le programme cette valeur sous format de chaine de caractères !... Il faut donc faire un cast de type quand vous lisez un entier (int) ou un réel (float). Exemple :

```
Annee_naissance = input(''entrer votre année de naissance '')
Annee_naissance = int(Annee_naissance)
```

Ou plus simplement en une seule instruction :

```
Anne_naissance = int(input(''entrez votre annee de naissance ''))
```

Exercice 1 : écrire un programme qui demande le nom xxx , l'année de naissance, et affiche par exemple bonjour xxx vous avez 20 ans… (Pour cela, vous avez besoin d'une variable qui contient l'année courante…)

Exercice 2 : demander un nombre n, doubler ce nombre, ajouter 10, diviser par 2, retrancher le nombre de départ, et afficher le résultat : vous devez trouver 5…

Exercice 3 : demander à l'utilisateur la valeur du rayon d'un cercle et calculer puis afficher le périmètre et la surface de ce cercle.

Exercice 4 : demander à l'utilisateur un nombre total de secondes, et en déduire le nombre correspondant d'heures, de minutes et de secondes…

Remarque : un commentaire se fait en commençant par #

IV. Les tests

Pour l'instant, nous avons vu des programmes où les instructions s'exécutent les unes après les autres pour donner un ou plusieurs résultats. Néanmoins, il est possible de prévoir des embranchements, à l'aide de tests, où certains blocs sont exécutés en fonction de la validité d'une ou plusieurs conditions. Un test se réalise à l'aide des mots clefs suivant :

```
if condition(s) :
        Bloc d'instructions 1
else :
        bloc d'instructions 2
```

```
if condition(s) :
        bloc d'instructions 1
elif condition(s) :
        bloc d'instructions 2
else :
        bloc d'instructions 3
```

Attention !....

*. Premièrement ne pas oublier les deux points ":" qui marquent le début d'un bloc

*. Deuxièmement : décaler chaque bloc dans les if ou elif ou else

*. Quand on revient à l'alignement du if ou elif ou else on sort d'un bloc

*. Le else est facultatif,

*. Vous pouvez imbriquer des tests bien sûr

*. Vous pouvez mettre plusieurs sous-tests commençant par elif, mais attention à l'ordre dans lequel vous placez les conditions correspondantes. Par exemple, si on cherche à faire un traitement différencié en fonction d'une variable x qui balaie un intervalle : en dessous de 0, entre 0 et 2, entre 2 et 4, et au-dessus de 4, la version de gauche est mauvaise (le cas x compris entre 0 et 2 ne sera jamais traité), la version de droite est correcte :

```
if x < 0 :
        …
elif x < 4 :
        …
elif x < 2 :
        …
else :
        …
```

```
if x < 0 :
        …
elif x < 2 :
        …
elif x < 4 :
        …
else :
        …
```

Les opérateurs de comparaison, sont <, <=, >, >=, != (différence) et == (égalité, attention, il faut mettre deux signes = pour ne pas confondre avec l'affectation d'une valeur à une variable)

Par ailleurs, vous pouvez combiner des conditions à l'aide des opérateurs logiques **and** ou **or** :

```
if x < 0 or x > 10 :
    print(''x en dehors  intervalle 0-10'')
```

```
if x > 0 and x < 10 :
    print(''x dans l intervalle 0-10'')
```

and : pour que l'ensemble soit vrai, il faut que les deux conditions soient vraies en même temps,

or : pour que l'ensemble soit vrai il suffit que l'une des deux soient vraie.

Pour terminer, vous pouvez utiliser aussi l'opérateur **not** qui retourne le booléen opposé : par exemple not True retourne False, not False retourne True. Vous pouvez écrire par exemple : if not x < 0 : …

Exercice 5 : prévoir un programme qui demande un nombre à l'utilisateur et qui à l'aide de tests calcule puis affiche sa valeur absolue

Exercice 6 : soit l'équation du seconde degré $ax^2 + bx + c = 0$. Demander à l'utilisateur a, b et c, puis calculer les racines (prévoir plusieurs cas…)

V. Les boucles

Les boucles son très importantes et permettent de répéter des opérations en quelques lignes de code. Plusieurs types de boucle existent en python : la boucle **for i in range(val_i, val_f)** qui donne une valeur à i entre val_i et val_f (attention !... val_f est exclue !... Si val_i n'est pas précisée alors on commence à 0 par défaut) et la boucle **while condition**, qui s'exécute tant que la condition est vraie (il faut donc modifier les variables intervenant dans la condition dans la boucle sinon on risque d'avoir une boucle infinie !...).

Par exemple les deux morceaux de code suivants affichent chacun les entiers i compris entre 1 et 4.

```
for i in range(1 ,5) :
    print(i)
```

```
i = 1
while i < 5 :
    print(i)
    i = i + 1
```

attention, comme pour les tests :

*. Ne pas oublier les deux points '' : ''

*. Décaler à l'intérieur du bloc

*. Quand on veut sortir du bloc, on écrit de nouveau en se positionnant en dessous du for ou du while…

D'une manière générale, il est possible d'imbriquer des boucles. Si par exemple, on a deux boucles, la boucle la plus interne est la plus rapide, et la boucle la plus externe est la plus lente… Par exemple si on veut afficher tous les couples i,j compris entre 1 et 10, on peut écrire :

```
for i in range(1,11) :
        for j in range(1,11) :
                print(i,j)
```

ou

```
i = 1
while i < 11 :
        j = 1
        while j < 11 :
                print (i,j)
                j = j + 1
        i = i + 1
```

Noter l'indentation (le décalage) entre les instructions, dans le cas du while… l'instruction i = i + 1 par exemple doit se trouver dans la boucle sur i mais pas dans la boucle sur j. D'une manière générale, ces deux boucles vont conduire à afficher pour chaque i (boucle externe) tous les j (boucle interne), nous aurons donc dans l'ordre à l'écran : (1,1), (1,2), …, (1,10), puis (2,1), (2,2),…, (2,10), (3,1), (3,2),…etc…

Dans les exercices suivants entraînez-vous avec les deux types de boucles…

Exercice 7.1 : pour un entier N demandé à l'utilisateur, calculer la somme des entiers de 1 à N… (Noter qu'à chaque itération sur i allant de 1 à N, on ajoute i à la somme s précédente, ce qui se traduit par s = s + i)

Exercice 7.2 : pour un entier N demandé à l'utilisateur calculer factorielle de n !

Exercice 8 : pour un entier N demandé à l'utilisateur, calculer la somme des entiers impairs compris entre 1 et N.

Exercice 9 : pour un entier N, déterminer tous les entiers i entre 2 et racine carree de N (int(sqrt(N)), qui divise N (un entier i divise N si le reste de la division entière de N par i est nul donc si N%i == 0). Ajouter un compteur « c » pour savoir le nombre d'entiers qui divisent N. Si « c » est égal à 0, alors afficher « nombre premier », sinon, afficher « nombre composé ».

Exercice 10 : demander à l'utilisateur un entier « total » compris entre 3 et 18 à l'aide d'une lecture contrôlée (c'est-à-dire qu'on redemande l'entier s'il n'est pas compris entre 3 et 18). Puis, simuler le lancement de trois dés à six faces en regardant toutes les combinaisons possibles, pour afficher celles dont la somme des trois dés est égale à « total ». Améliorer le programme en évitant d'afficher les permutations (par exemple pour un total égal à 5, on veut éviter d'afficher 1,1,3, puis 1,3,1, puis 3,1,1, mais seulement 1,1,3).

Exercice 11 : nous allons procéder au calcul d'une valeur approchée de pi par l'utilisation de tirages aléatoires. Soit un carré de coté 1, qui contient un quart de cercle de rayon 1. La surface sc du carré est égal à 1*1, et la surface sd du quart de disque est égale à Pi*r**2/4. Le rapport sd/sc est donc égal à Pi/4. Maintenant, nous allons de manière répétée tirer un point avec deux nombres alétoires x et y

compris entre 0 et 1. La variable n contiendra le nombre total de points tirés, et la variable p contiendra le nombre de points qui se trouvent dans le quart de disque (donc avec sqrt(x**2+y**2) <= 1.). Le rapport p/n va traduire le rapport des deux surfaces, donc p/n = Pi/4. Aussi, on en déduit Pi = 4p/n. Ecrire le programme correspondant, vous pourrez par exemple procéder à k tirages, avec k fixé par l'utilisateur, et vous afficher pi, ou bien vous pouvez continuer les tirages tant que la variation sur la valeur de pi est supérieure à une certaine précision epsilon. Afin de tirer un nombre aléatoire en python :

```
from random import *
```

et utiliser la fonction random() qui retourne un réel compris entre 0 et 1 (par exemple : x = random()).

VI. Les listes

Pour l'instant, nous utilisions des variables pour stocker et manipuler de l'information. Les listes permettent de stocker plusieurs informations d'un coup. Une liste est notée entre crochets. **Une liste vide** nommée L par exemple s'écrit : L = [ ]. Nous pouvons mettre plusieurs éléments (éventuellement identiques) de type différent dans une liste, ou bien que des entiers, ou que des réels par exemple. Voici quelques listes :

L1 = [''benoit '', 1970, 1.8]

L2 = [10, 3, 1, 5, 7, 3, 6]

L3 = [2.3, 5.2, 4.3]

Ces trois instructions permettent d'initialiser les trois listes L1, L2 et L3. Mais parfois on veut ajouter un élément x à une liste L, on utilise alors **L.append(x).** Par exemple, L3.append(7.9) ajoute le réel 7.9 à la fin de la liste L3, donc la liste L3 contient désormais [2.3, 5.2, 4.3, 7.9]. Si l'on veut accéder à l'élément i de la iste L, on utilisera L[i]. Attention, i est l'indice, et cet indice commence à 0 et va jusqu'à n-1 pour une liste de n éléments. Ainsi, dans L2 qui contient 7 éléments, L2[0] correspond à la valeur 10, L2[1] à la valeur 3. Pour savoir le nombre d'éléments d'une liste, on utilise l'instruction **len(L)**.

Par exemple, une fois L3 initialisée, si nous voulons afficher tous les éléments de la liste L3, avec leur indice, nous pouvons parcourir chacun des éléments et les afficher, donc utiliser une boucle :

```
i = 0
while i < len(L3) :
        print(i, L3[i])
        i = i + 1
```

ou

```
for i in range(0,len(L3)) :
        print(i, L3[i])
```

Pour créer une liste L de n éléments telle que L[i] = i, nous pouvons écrire :

```
L = [ ]
N = int(input(''entrer n ? ''))
i = 0
while i < N :
        L.append(i)
        i = i + 1
```

Les listes que nous venons de voir sont à une dimension (besoin d'un seul indice pour accéder aux éléments), mais il est possible de créer des listes à deux dimensions. Par exemple la liste L4 de 3*3 éléments:

L4 = [[1,2,3], [4,5,6], [7,8,9]]

Et pour accéder à un élément, on utilise L4[i][j]. Ainsi L4[0][0] vaut 1, L4[1][2] vaut 6…

Imaginons que l'on veuille créer une liste L5 qui représente une matrice N par M, avec que des 0, nous pourrions écrire :

```
N = int(input(‘’entrer N : ‘’))
M = int(input(‘’entrer M : ‘’))
L = [ ]
j = 0
while j < M :
        L.append(0)          # on ajoute des 0 dans la liste intermédiaire L
        j = j + 1
L5 = [ ]
i = 0
while i < N :
        L5.append(L)         #on ajoute la liste L comme élément de liste dans L5
        i = i + 1
```

Exercice 12.a : définir par une boucle une liste qui contient tous les entiers pairs compris entre 0 et N (demandé à l'utilisateur), puis afficher le contenu de cette liste.

Exercice 12.b : donner les instructions permettant de calculer le maximum d'une liste

Exercice 13 : Définir les instructions permettant de voir si une liste est symétrique (vous pouvez tester en initialisant des listes vous-même…).

Exercice 14 : définir les instructions permettant de renverser une liste. Le premier élément devient le dernier, le second l'avant dernier, etc…

Exercice 15 : écrire le programme permettant de calculer la moyenne des éléments d'une liste.

VII. Les fonctions

Les fonctions permettent de découper le programme en regroupant des instructions en sous-bloc, et peuvent être appelées dans d'autres fonctions ou dans la partie principale du programme. Ainsi, plutôt que d'écrire les instructions d'affichage pour chaque liste manipulée, qui suppose une boucle, il suffit de regrouper ces instructions dans une fonction affichage(L) qui reçoit en argument une liste L, et de l'appeler à chaque fois que l'on en a besoin.

Pour définit une liste on utilise le mot-clef **def** selon le schéma suivant :

```
def nom_fonction(argument(s)) :
        Bloc d'instructions
```

Exemple :

```
# definition de la fonction affichage
def affichage(L) :
        for i in range (len(L) :
                print(L[i])

# 2 appels de la fonction affichage
Affichage(L2)
Affichage(L3)
```

Cette fonction affichage ne retourne pas de résultat, mais bien souvent on veut récupérer le résultat d'un calcul effectué dans la fonction. On utilise alors l'instruction **return x**. Par exemple, la fonction minimum(a,b) retourne la valeur la plus petite entre a et b reçus en arguments de la fonction :

```
# definition de la fonction minimum
def minimum (a, b) :
        if a < b :
                return a
        else :
                return b

# deux appels de la fonction minimum
X = minimum(5.7, 2.3)
Y = minimum(1, 8)
```

Parfois, on veut récupérer plusieurs résultats. Pour ce faire on utilise les tuples qui sont des groupes de données entre parenthèses : par exemple le tuple doublet contenant a et b sera noté (a,b). Soit la fonction somme_produit(x,y) qui calcule la somme de x et y, et le produit x*y. A l'appel, on décompose directement le tuple résultat en utilisant des variables séparées par des virgules.

```
# definition fonction somme_produit
def somme_produit(x,y) :
        return (x+y, x*y)

# appels de la function
s1, p1 = somme_produit(3,8)
s2, p2 = somme_produit(2.2, 7.5)
```

Exercice 16 : Ecrire une fonction permut(a,b) telle que a prend la valeur de b, et b celle de a. Appeler cette fonction et imprimer les valeurs de a et b avant et après l'appel. Qu'observez-vous ?...

Exercice 17 : Ecrire une fonction init(L,n) qui initialise L avec tous les nombres impairs compris entre 3 et n. Après l'appel de la fonction la liste L contient-elle bien les valeurs de ces entiers impairs ?.. Qu'en déduire sur la mise à jour en mémoire de L dans la fonction ?...

Exercice 18 : écrire la fonction divise(n,m) qui retourne vrai si m divise n. Ensuite écrire une fonction liste_premier(n) qui reçoit un entier n, et crée la liste des entiers 2 < = i < = n qui sont premiers. Ecrire

une troisième fonction qui affiche la liste L des nombres premiers. Appeler la ou les fonctions après avoir demandé n à l'utilisateur.

Exercice 19 : écrire une fonction moyenne(L) qui calcule la moyenne des éléments de la liste L et retourne cette valeur. Puis écrire la fonction variance qui calcule la variance c'est-à-dire la moyenne des écarts au carré : var = 1/n somme (L[i]-moyenne(L))**2. Vous pouvez utiliser une liste intermédiaire des écarts au carrés par exemple. Tester vos fonctions sur quelques listes…

VIII. Pour aller plus loin, les fichiers

Afin de lire ou d'écrire dans des fichiers, ce qui est très commode pour faire des visualisations ultérieures par exemple, ou pour relancer des calculs. Pour ce faire, on utilise les instructions **open**, **close**, **readline**, **split**, **write**… Par exemple, pour lire le fichier « notes.dat » suivant qui contient des informations sur le nombre d'étudiants (3) puis les étudiants et leurs notes:

3

Paul 16

Emilie 12

Sophie 18

Nous pourrions écrire en lecture :

```
f_in = open(''notes.dat'', ''r'')     # ouverture du fichier f_in, r pour read, w pour write
nb_etu = int(f_in.readline())
s = 0.0
i = 0
While i < nb_etu :
      str = f_in.readline()     # lecture 1 ligne du fichier f_in stockée
                                # dans la chaine de caractères str
      L = str.split             # separation de str stockee dans une liste L
      s = s + float(L[1])       # somme de la note transformée en réel
      i = i + 1

s = s / nb_etu
print('' moyenne des notes = '', s)
f_in.close()  # fermeture du fichier
```

Pour écrire dans un fichier, il faut utiliser ''w'' à la place de ''r'' dans le open, et utiliser l'instruction write. Attention write écrit seulement des chaines de caractères, si vous manipulez des entiers ou des réels, il faut utiliser str(x) qui transforme x en chaine de caractères.