ALGORITHMIQUE ET PROGRAMMATION

I Un exemple d'algorithme égyptien: Ahmès

On donne les deux nombres 50 et 64.

1. Écrire une colonne d'entiers sous le nombre 50 dont chaque nombre est la moitié, ou la plus petite des deux moitiés s'il y a le choix, du nombre situé immédiatement au-dessus de lui, et en s'arrêtant au nombre 1.
2. Écrire à la droite de la colonne précédente, une colonne commençant au nombre 64, de longueur égale à celle de la précédente colonne, et où chaque nombre est le double du nombre situé immédiatement au-dessus de lui.
3. Écrire, à la droite de la colonne précédente, une troisième colonne où, pour chaque ligne, si le contenu de la première colonne est impair, on recopie le contenu de la seconde colonne, sinon on laisse en blanc.
4. Additionner les nombres de la troisième colonne.

50	64

II Le même en « langage naturel », puis en langages de programmations.

Langage naturel :

Algorithme d'Ahmès

En Python :

```
def ahmes ( a , b) :
    x = a
    y = b
    z = 0
    while x != 0 :
        if x%2 == 1 :
            z = z+y
        x = int ( x /2)
        y = y*2
    return z

c = int(input("Donnez un premier entier :")
d = int(input("Donnez un deuxième entier :")
print("Résultat :", ahmes(c,d))
```

En C :

```
#include <stdio.h>
int ahmes(int a, int b) {
        int x = a ;
        int y = b ;
        int z = 0 ;
        while (x != 0) {
            if (2*(x/2) != x)
                z = z+y ;
            x = x/2 ;
            y = y*2 ;
        }
}
main() {
            printf("%d", ahmes(50,64)) ;
        }
```

III Rappels (en théorie) sur l'algorithmique.

Les instructions autorisées sont :

- Affectation de variables (symbolisées par ←, →, :=, voire =)
- Instruction répétitive :
  - Tant que *condition* faire… fin tant que.
  - Instructions complémentaires
    Pour *variable de valeur de début à valeur de fin* faire… fin pour
    Répéter… jusqu'à *condition*
- Instruction conditionnelle :
  - Si *condition* alors… (sinon…) fin si
  - Instruction complémentaire (aiguillage traduit en switch, case, elif):
    Si *condition1* alors … sinon si *condition2* alors… etc… (sinon…) fin aiguillage
- Entrée/sortie :
  - Lire au clavier
  - Afficher/imprimer à l'écran

Pour chaque conditionnelle, chaque répétitive, les instructions doivent être décalées, et un trait doit limiter la séquence d'instructions à effectuer à l'intérieur de cette conditionnelle/répétitive.

IV Quelques notions de base sur la programmation.

1. Qu'est-ce que programmer ?
   Programmer, c'est résoudre des problèmes, puis les traduire en une suite d'ordres donnés à l'ordinateur.
   Un ordinateur sans programme ne sait rien faire : *Windows* est un programme, les jeux sont des programmes.
2. Les différents types de langage de programmation
   Le seul langage de programmation directement utilisable par un ordinateur est le langage machine, sous forme binaire (formé uniquement de 0 et de 1).
   Personne ne programme en langage machine car c'est trop compliqué. Ainsi, les informaticiens ont inventé de nombreux langages qui utilisent des instructions au lieu d'une suite de 0 et de 1. Ces instructions, une fois écrites par le programmeur, sont traduites en langage machine, à l'aide d'un programme destiné à cet effet. Ce système de traduction s'appelle interpréteur ou bien compilateur, suivant la méthode utilisée pour effectuer la traduction.
   Il existe 2 types de langages de programmation :
   - les langages de bas niveau : très complexes à utiliser (car très éloignés du langage naturel), on dit que ce sont des langages « proches de la machine ». Ils permettent en contrepartie de faire des programmes très rapides à l'exécution.
   - Les langages de haut niveau : ils sont plus faciles à utiliser car plus proches du langage naturel.
3. Compilation et interprétation
   Le programme tel que nous l'écrivons est appelé programme source (ou code source). Comme déjà signalé plus haut, il existe deux techniques principales pour effectuer la traduction d'un programme source en langage machine : l'interprétation et la compilation.
   a *L'interprétation*
   Dans cette technique, le logiciel interpréteur doit être utilisé chaque fois que l'on veut faire fonctionner le programme. Chaque ligne du programme source analysé est traduite au fur et à mesure en quelques instructions du langage machine, qui sont ensuite directement exécutées. Python, Javascript, par exemple, sont des langages interprétés.
   b *La compilation*
   La compilation consiste à traduire la totalité du code source en une seule fois. Le logiciel compilateur lit toutes les lignes du programme source et produit une nouvelle suite de codes, écrits en langage machine, que l'on appelle programme objet (ou code objet). Celui-ci peut désormais être exécuté indépendamment du compilateur et être conservé tel quel dans un fichier appelé fichier exécutable. Le *C++*, par exemple, est un langage compilé.

c *Avantages et inconvénients de l'interprétation*

L'interprétation est idéale lorsque l'on est en phase d'apprentissage d'un langage, ou en cours d'expérimentation sur un projet. Avec cette technique, on peut en effet tester immédiatement toute modification apportée au programme source, sans passer par une longue phase de compilation qui peut durer des heures pour de très gros programmes.

Par contre, traduire les instructions à la volée ralentit l'exécution du programme. Les programmes en langage interprété sont toujours plus lents à l'exécution que les programmes en langage compilé.

d *Choix du langage Python*

Il existe un très grand nombre de langages de programmation, chacun ayant ses avantages et ses inconvénients.

Parmi les langages libres[1] et gratuits il existe des interpréteurs et des compilateurs pour toute une série de langages, mais surtout ces langages sont modernes, performants, portables (c'est-à-dire utilisables sur différents systèmes d'exploitation tels que *Windows*, *Linux*, *Mac OS* ...), et fort bien documentés.

Le langage dominant est sans conteste le *C/C++* et ses variantes. Ce langage s'impose comme une référence absolue, et tout informaticien sérieux doit s'y frotter tôt ou tard. Il est assez proche de la machine donc sa syntaxe est peu lisible (la remarque sur la lisibilité vaut aussi dans une large mesure pour le langage *Java*, très répandu également).

Pour les débuts dans l'étude de la programmation, il est préférable d'utiliser un langage de plus haut niveau, moins contraignant, à la syntaxe plus lisible. Nous avons décidé d'adopter *Python*, langage couramment utilisé.

V Comment utiliser Python

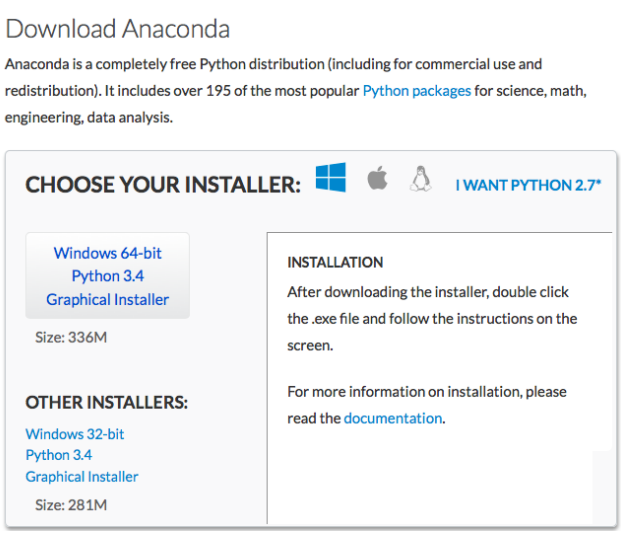
Les outils dont vous aurez besoin peuvent se limiter à Python et un environnement de développement. Dans ce cas seul vous n'avez qu'à suivre les instructions du 1 ; quelques compléments sont donnés dans le 3. Si vous souhaitez reprendre les TD sur iPython notebook, il vous faut également suivre les instructions du 2.

1. Installation de Python (pour : exercices, devoirs, projet)

   La première chose à faire pour utiliser Python est de l'installer, si nécessaire…

   Si vous utilisez *Linux* sur votre machine, vous avez peut-être déjà *Python 3*. La plupart des distributions *GNU/Linux,* ainsi que Mac OsX, installent par défaut *Python 2*, de plus en plus de distributions fournissent également *Python 3*. Les autres systèmes d'exploitation comme Windows ne sont fournis avec aucune version de Python.

   Si Python est installé sur votre machine, assurez vous qu'il s'agit de la version 3. Si tel n'est pas le cas, vous pouvez downloader Python à : http://www.python.org/download/. Veillez à télécharger la dernière version (Python 3) correspondant à votre environnement (*Windows*, *Macintosh*, etc.).



2. Installation de iPython (pour reprendre les cours en notebook).

   Si vous souhaitez retravailler les cours interactifs distribués chez vous, il est indispensable d'utiliser iPython et iPython notebook.

   Vous pouvez utiliser un environnement Python temporaire avec http://tmpnb.org. Inconvénient majeur : l'upload est très capricieux.

   Deuxième méthode, quasiment aussi simple, et bien plus fiable : télécharger la distribution Anaconda et

[1] Un **logiciel libre** (*Free Software*) est avant tout un logiciel dont le code source est accessible à tous (*Open Source*). Souvent gratuit (ou presque), copiable et modifiable librement au gré de son acquéreur, il est généralement le produit de la collaboration bénévole de centaines de développeurs dispersés dans le monde entier. Des exemples de logiciels libres : le système d'exploitation *GNU/Linux*, la suite O*pen Office*. Un logiciel non libre est dit **propriétaire**. Des exemples de logiciels propriétaires : le système d'exploitation *Windows*, la suite *Microsoft Office.*

son Navigator à https://www.anaconda.com/download/ . Choisir la version correspondante à votre système d'exploitation.
Puis « upload » le fichier à utiliser, et double-clic pour le lancer.

Troisième méthode, plus complexe, parce que l'on utilise des commandes en ligne, et non une interface graphique. Mais on y survit !

- Sous OsX ou Linux, c'est faisable en quelques étapes, en commençant par :
  - Ouvrir un terminal (OsX : applications>utilitaires, Linux : dépend de l'interface graphique utilisée). Sauter la partie ci-dessous « sous Windows ».
- Sous Windows, le début est un peu plus compliqué (n'ayant pas Windows8, je ne l'ai fait que sous Windows7, c'est donc à adapter).
  - Après avoir installé Python, il faut changer les variables d'environnement (la méthode provient de https://adesquared.wordpress.com/2013/07/07/setting-up-python-and-easy_install-on-windows-7/) :
    - dans le panneau de démarrage, clic droit sur ordinateur>propriétés
    - dans la fenêtre qui s'ouvre, paramètres systèmes avancés>paramètres systèmes avancés>variables d'environnement
    - chercher "path" dans variable système, rajouter à la fin ;C:\Python34;C:\Python34\Scripts (si après ça ne marche pas vérifier dans le lien ci-dessus si c'est vraiment ça)
  - Ouvrir une invite de commande en tant qu'administrateur (avec un clic droit sur invite de commande, dans menu démarrer>accessoires)
- Puis, pour tous les systèmes :
  - Taper « pip install ipython » après l'invite de commande (le premier caractère en début de ligne, un $ souvent). Le terminal/l'invite de commande vous raconte tout un tas de choses, puis :
  - Taper « pip install pyzmq jinja2 tornado mistune jsonschema pygments terminado »
  - Sur certains systèmes, il semblerait qu'il faille aussi faire « pip install notebook »
  - Remarque : vous pouvez faire un copier coller de l'instruction précédente directement à partir de https://ipython.org/ipython-doc/3/install/install.html#installation-using-pip, dans le paragraphe « dependancies for the ipython HTML notebook »

3. L'environnement de développement (IDE)

   Avant de commencer à écrire des programmes *Python* dans des fichiers sources, nous avons besoin d'un éditeur de texte. Le choix de l'éditeur est très important. L'un des besoins de base est la coloration syntaxique. Ainsi les différentes parties de votre programme sont coloriées de différentes couleurs afin d'en faciliter la lecture et d'éviter les erreurs.

   Pour votre apprentissage, nous avons opté pour l'éditeur de texte *IDLE*. En plus de la coloration syntaxique, *IDLE* offre la possibilité de lancer votre programme à l'intérieur de *IDLE*. Celui-ci est installé par défaut avec les installeurs de *Python* pour *Windows* et *Mac OS X*. Il est aussi disponible à l'installation pour *Linux*.

   Dans un deuxième temps, vous pouvez utiliser une distribution de Python plus complexe, par exemple Pyzo, qui est très bien faite (http://www.pyzo.org/downloads.html). L'avantage est d'avoir le terminal et le code source dans une même fenêtre. Vous pouvez aussi utiliser Spyder, sur Ananconda Navigator. Il arrive que la version de Python utilisé dans Spyder ne soit pas la dernière, car SPyder possède en plus des bibliothèques scientifiques spécialisées (que vous utiliserez pour ceux qui iront en prépa).

   Il existe des environnements de développement bien plus complexes, comme Eclipse par exemple. Ceci au cas où vous souhaiteriez programmer le prochain Call Of Duty tout(e) seul(e).

   Vous pouvez lancer IDLE en allant dans le Menu démarrer > Tous les programmes > Python 3 > IDLE (Python GUI), ou bien dans MCNL.

4. Un manuel

   Qui offre l'avantage d'être gratuit (en téléchargement légal), assez complet, et pas mal fichu, malgré un index parfois défaillant. Il s'agit de G.Swinnen : Apprendre à programmer avec Python 3.

   Téléchargeable gratuitement à cette adresse : http://inforef.be/swi/python.htm.

5. Remarque sur les noms de fichier

   Je vous conseille, lorsque vous faites un exercice, de mettre comme nom de fichier :
   TD1_Ex1_VotrePrénom.py, et de mettre à nouveau en première ligne de commentaire le numéro de l'exercice, ainsi que votre prénom.

VI Les bases du Python.

1. Commentaires.

   Les commentaires sur une ligne, ou en fin de ligne, commencent par # ; sont indispensables pour la compréhension, la réutilisation ultérieure et la relecture d'un programme.

   On peut mettre des commentaires de plusieurs lignes entre triples guillemets : « « « … » » »

   Pour des raisons de compatibilité sur les différents systèmes, évitez les accents ainsi que tous les caractères spéciaux.

2. Expressions.

Outre les opérations arihtmétiques standard, on dispose des opérateurs suivants:

- //
- %
- **

3. Variables

Le nom des variables suit quelques règles de base :

- un nom de variable est une suite de lettres (a → z , A → Z) et/ou de chiffres (0 → 9), qui doit toujours commencer par une lettre ;
- seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont à proscrire, à l'exception du caractère _ (tiret bas ou underscore) ;
- la casse est significative (les caractères majuscules et minuscules sont distingués). Par exemple, Nombre, nombre, NOMBRE sont des variables différentes. Soyez attentifs !
- 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 nomDeFamille[2]. Par contre, si vous utilisez le nombre π souvent, c'est une constante ; on écrira alors la variable correspondante en majuscules PI = 3.14159.
- N'hésitez pas à donner des noms clairs à vos variables. Par exemple, nomDeFamille est plus clair que nf.
- De plus, les variables ne peuvent pas être un des « mots réservés[3] » suivant utilisés par le langage lui-même :

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

4. Booléens

Les booléens sont des variables qui ne prennent que deux valeurs : vrai ou faux (True et False respectivement en Python, avec des majuscules)

Les variables booléennes sont utilisées pour les tests lors des boucles ou des conditionnelles. On peut les combiner à l'aide des connecteurs logiques : et, ou, non qui sont en Python and, or, not.

Opérateurs de comparaison :

```
x == y          # x est égal à y
x != y          # x est différent de y
x > y      # x est plus grand que y
x < y      # x est plus petit que y
x >= y          # x est plus grand que, ou égal à y
x <= y          # x est plus petit que, ou égal à y
```

Attention : pas d'encadrement en programmation

Opérateurs logiques :

```
a and b         # « et » mathématique
a or b          # « ou » mathématique
not(a)          # « non » mathématique
```

Ils correspondent aux notions vues en probabilité sur les événements.

[2] Cette convention est très utilisée. Personnellement, je trouve nom_de_famille plus lisible.

[3] La liste est variable suivant les versions de Python

5. Instructions conditionnelles

Le « si *condition* alors *instruction* (sinon *instruction*) » se traduit par :

```
if condition :
    instruction 1
(else :)
    instruction 2
suite du programme
```

Vous remarquez l'indentation, c'est à dire le fait que l'instruction est décalée vers la droite (en général de 4 espaces). Ceci traduit le fait que l'instruction 1 sera exécutée uniquement si la condition dans le « si » est vraie, l'instruction 2 étant exécutée uniquement si la condition est fausse. Bien sûr, l'instruction 1 peut être formée de plusieurs instructions élémentaires. Si l'on imbrique plusieurs « si », alors on indentera à chaque fois les instructions. N'oubliez pas les « : ».
L'ensemble if *condition* : *instruction 1* forme une instruction composée.

La condition évalue un booléen, cf TD2.

L'instruction « `if`…`elif`…`elif`…etc…(`else :`) » permet de choisir entre plusieurs alternatives (elif est la contraction de else if)

6. Instructions répétitives

Le « tant que » est traduit par :

```
while condition :
    instruction
suite du programme
```

Même remarque sur l'indentation, ainsi que sur la condition qui est un booléen, qu'au paragraphe précédent

Le « pour » est traduit par :

```
for variable in séquence :
    instruction 1
suite du programme
```

Par exemple, « `for i in range(0, 10, 3) :` » exécutera l'instruction 1 pour les valeurs de *i* respectivement égales à 0, 3, 6, 9. Si on écrit `range(0, 10)` , alors *i* prend toutes les valeurs de 0 à 9 (10 exclu).

Lorsque l'on a une condition compliquée à tester pour sortir d'une boucle par exemple, il est souvent pratique d'utiliser un booléen (exemple : un jeu que l'on peut finir de plusieurs manières à l'intérieur de la boucle, cf. exercice « deviner un nombre »). Dans ce cas, si « `continuer` » est le nom de la variable booléenne, on écrit « `while continuer` », et non « `while continuer == True` » qui est un pléonasme informatique !

7. Modules

Une bibliothèque (ou un module) est un ensemble de fonctions préprogrammées, ainsi le programmeur n'a pas à les refaire.
On inclut une bibliothèque dans un programme à l'aide de la commande « import ».
Nous utiliserons les bibliothèque « math », qui permet de calculer des cosinus, des racines carrées, des valeurs absolues… et la bibliothèque « random », qui permet d'obtenir des nombres aléatoires. Ultérieurement, nous utiliserons également des bibliothèques graphiques pour les interfaces.
On tape si nécessaire en début de programme :

```
>>> from math import *
>>> from random import *
```

8. Lisibilité des programmes et fonctions

Pour être lisible, un programme doit être structuré en fonctions, que l'on écrira au début du programme.

Lorsque l'on réfléchit à un problème que l'on veut programmer, il faut le faire en termes de fonctions : « je veux d'abord une fonction qui me permet de rentrer mes données », « je veux une fonction qui teste si une variable est positive et dans ce cas calcule sa racine carrée » etc…
La syntaxe pour les fonctions est :

`def` *nom_de_fonction* (*variables passées en paramètre, à utiliser dans la fonction*) :
    instructions
    `return` (*variable*)

L'appel dans le programme se fait par :
variable_resultat = nom_de_fonction (*variables à passer en paramètre*)

Exemple : le programme suivant teste vos capacités en calcul mental

```
# F. Mandon
#
# Programme de test de connaissance sur les tables de multiplication

# bibliotheques
from random import *

########################
#
# Fonctions
#
########################

# Fonction de choix d'un nombre aléatoire
#   donnees : aucune
#   resultat : renvoie un nombre aléatoire entre 2 et 10

def alea10():
    return (randint(2,10))

# Fonction de test du produit
#   donnees : trois entiers x, y, z
#   resultat : un booleen gagne, qui vaut vrai si c = a*b et faux sinon

def testTable(x,y,z):
    if z == x*y:
        gagne = True
    else:
        gagne = False
    return(gagne)

######################
#
#   Programme principal
#
####################

a = alea10()
b = alea10()
print("Que vaut le produit de ",a," par ",b," ?")
c = int(input())
juste = testTable(a,b,c)
if juste:
    print("Bravo !")
else:
    print("Lamentable...")
```

Vous remarquez que les fonctions sont commentées, on indique d'abord leur nom, puis le type des

variables passées en paramètre (avec d'éventuelles restrictions), et enfin le type de la variable renvoyée. On peut préciser dans les commentaires la « stratégie » de la fonction, si elle est un peu compliquée. Ces commentaires sont indispensables : la notation se fait en partie non négligeable dessus. Il est préférable de taper les commentaires (ici appelés *spécifications*) au fur et à mesure, pendant que l'on sait ce que l'on vient de faire…

9. Tableaux/listes.

Sous Python, une liste est une vairable contenant plusieurs variables, représentées par une suite d'éléments separés par des virgules, le tout étant entre crochets.

Ex :

```
>>> matieres =['maths', 'philo', 'svt', 1800, 20.357, 'physique', 'cuisine']
>>> print(matieres)
['maths', 'philo', 'svt', 1800, 20.357, 'physique', 'cuisine']
>>> listeVide = []
>>> listeDeZeros = [0]*5
>>> print(listeDeZeros)
[0,0,0,0,0]
```

Vous pouvez constater que les éléments (items) peuvent être de type distinct.

Une liste est indexée (ou indicée). C'est à dire que l'on accède à un item précis d'une liste comme suit (attention la numérotation commence à 0) :

```
>>> print(matiere[0])
maths
```

Si on veut les éléments de la deuxième à la quatrième position, on tapera :

```
>>> print(matieres[1:4])
   ['philo', 'svt', 1800]
```

Dans l'instruction précédente, remarquez que, comme pour `range`, l'extraction des index se fait avec un intervalle fermé à gauche et ouvert à droite.

Des indexs négatifs peuvent aussi être utilisés, -1 désignant le dernier item de la suite.

```
>>> print(matieres[-3 :]) # idem : print(matieres[-3 :-1])
[20.357, 'physique', 'cuisine']
```

Remarque : attention au manque de cohérence dans les intervalle d'extraction d'index. `[1:4]` récupére les index `1`, `2` et `3`, alors que `[-4 :-1]` récupère `-4`, `-3`, `-2` et `-1`…

La longueur d'une liste est donnée par len(*nom_de_la_liste*) :

```
>>> len(matieres)          # renvoie 7
```

On peut supprimer un ou plusieurs élément(s) d'une liste avec del() :

```
>>> del(matieres[3:5])
>>> print(matieres)
['maths', 'philo', 'svt', 'physique', 'cuisine']
```

Enfin, on peut ajouter un item en fin de liste avec la *méthode* append. Attention, ça n'est pas une fonction, la syntaxe est différente. En effet, on n'affecte pas le résultat dans une variable. Une *méthode* modifie directement l'*objet* sur lequel elle s'applique.

```
>>> matieres.append('starcraft')
>>> print(matieres)
      ['maths', 'philo', 'svt', 'physique', 'cuisine', 'starcraft']
```

Une fois les bases sur les listes maitrisées, on pourra utiliser les méthodes suivantes :

- `liste.sort()` : trie une liste dans l'ordre croissant (cf. doc pour l'ordre décroissant et les options)
- `liste.remove(valeur)` : supprime la première occurrence de `valeur` dans la liste
- `liste.insert(index, valeur)` : insère `valeur` à l'indice `index`
- `liste.reverse()` : inverse les éléments de la liste
- `liste.count(valeur)` : compte le nombre d'occurrences de `valeur` dans la liste (à récupérer dans une variable)
- `liste.index(valeur)` : donne l'index de la première occurrence de `valeur` dans la liste (à

récupérer dans une variable)

- `liste1.extend(liste2)` : rajoute en fin de `liste1` la `liste2`. Il est plus simple de faire `liste1 + liste2` !

L'instruction « in ».

L'instruction in permet de tester si une valeur est dans une liste ou non, et aussi de boucler sur une liste.

```
>>> 'maths' in matieres
True
>>> for item in matieres :
      print(item)
# renvoie :
'maths'
...
'starcraft'
```

Tableaux et matrices.

Un tableau est une liste de listes. La création d'un tableau peut être délicate, cf. exercices.
Une matrice est un tableau de nombres. Toutes les lignes doivent avoir la même longueur.
Pour accéder à un élément, on met d'abord l'index de ligne puis celui de colonne :

```
>>> mat3x2 = [[1,2],[3,4],[5,6]]
>>>for i in range(len(mat)):
        print (mat[i])

  [1, 2]
  [3, 4]
  [5, 6]
>>> print(mat3x2[1][0])
  3
```

10. Chaînes de caractères.

Un chaîne de caractère est une liste de caractères, d'un type particulier. Certaines des fonctions et méthodes précédentes sont donc utilisables.

Ex :

```
>>> profession = "informaticien"
>>> print(profession[0],profession[3]," longueur ", len(profession))
i o  longueur  13
```

On dispose d'autres opération, comme la *concaténation*.

```
>>> a = "j'aime "
>>> b = "pas "
>>> c = "les mathématiques"
>>> print(a+c)
>>> j'aime les mathématiques
```

Il existe de nombreuses autres méthodes sur les chaines, si nécessaire consulter la doc python.

EXERCICES PROGRAMMATION

Ex 1. On dispose de la formule suivante pour convertir les degrés Fahrenheit en degrés Celsius : $C = 0,55556 \times (F - 32)$, où F est une température en degrés Fahrenheit et C la température correspondante en degrés Celsius.

1. Ecrire un programme qui convertit en degrés Celsius une température rentrée au clavier en degrés Fahrenheit.
2. Même question pour la conversion inverse.

Ex 2. Écrire un programme qui permute et affiche les valeurs de trois variables *a*, *b*, *c* qui sont entrées au clavier : *a* ==> *b* , *b* ==> *c* , *c* ==> *a*.

Ex 3. Ecrire un programme qui donne le nombre de solutions réelles de l'équation du second degré, ainsi que leur valeur. Les coefficients *a*, *b* et *c* seront rentrés au clavier.

Ex 4. Ecrire un programme qui lit une valeur et affiche sa table de multiplication (on se limitera aux 12 premiers termes)

Faire une variante du programme précédent qui affiche la table de multiplication de tous les chiffres compris entre 2 et 9 (inclus).

Remarque : Pensez à laisser un espace entre deux tables de multiplication (`print()` imprime une ligne vide).

Ex 5. Écrire un programme qui affiche un triangle rempli d'étoiles (*) sur un nombre de lignes donné passé en paramètre, exemple :

```
*
**
***
****
*****
******
*******
********
```

- 1ère version : à l'aide de deux boucles for, en imprimant les * une par une. On remarquera que, par défaut dans l'instruction « print », figure end = '\n', qui fait passer à la ligne. print(…, end ='') ne fera donc pas passer à la ligne. De même, end = ' lol ' vous fera passer pour utilisateur standard de facebook
- 2ème version : avec une seul boucle for, et une chaîne de caractères où vous accumulerez des étoiles (pour ceux qui vont un peu plus vite, print(« machin » end= '') évite de passer à la ligne.

Ex 6. Ecrire un programme qui teste si un nombre *a* est divisible par un nombre *b*, les deux étant rentrés au clavier. Le programme retournera un message signalant la divisibilité ou non, et éventuellement le reste dans le cas où *a* n'est pas divisible par *b*.

Ex 7. Soit *f* la fonction définie sur $\mathbb{R}$ par $f(x) = (3x-1)(2x^2+1)$.

1. Ecrire un programme qui calcule l'image d'un nombre rentré au clavier.
2. Une approximation du nombre dérivé en *a* est donnée par $f'(a) = \dfrac{f(a+h) - f(a)}{h}$, pour *h* assez proche de 0. Écrire un programme qui calcule une approximation du nombre dérivé en un point *a*, rentré au clavier. On saisira également la valeur de *h*.
3. Pour ceux qui sont « avancés », écrire un programme donnant le tableau de valeurs comme sur la calculatrice, avec choix de la valeur de début, du pas, et du nombre total d'images.
4. Pour ceux qui sont « encore plus avancés », écrire un programme donnant le choix entre les deux opérations précédentes (calcul de l'image ou du nombre dérivé), et qui fait appel à deux fonctions, l'une pour l'image, l'autre pour le nombre dérivé.

Ex 8. Programmation d'un petit jeu de devinette. L'ordinateur choisit au hasard un nombre compris entre 1 et 100. Le but du jeu est de le deviner en un nombre d'essai minimal. À chaque tentative, l'ordinateur, indique « gagné », « trop petit » ou « trop grand ». L'utilisateur dispose d'un nombre d'essais limités.

Écrire l'algorithme en « langage naturel ». Programmer le jeu, et le tester.

Remarque : on utilisera la bibliothèque *random.*

Pour cela, on écrit « import random » en début de programme.

nombre = `random.randint`(*a*, *b*) renverra un nombre aléatoire tel que $a \leq Nombre \leq b$

Pour plus d'informations et de possibilités : http://docs.python.org/release/3.0.1/library/random.html

Ex 9. Nombres parfaits

Un nombre entier est parfait s'il est égal à la somme de ses diviseurs (sauf lui-même).

Ex : 6 = 1 + 2 + 3 est parfait.

Écrire une fonction somme_div qui retourne la somme des diviseurs d'un nombre passé en paramètre.

Écrire une fonction parfait qui teste si un nombre passé en paramètre est parfait et qui retourne 1 s'il l'est et 0 sinon. Écrire un programme principal qui affiche tous les nombres parfaits inférieurs à une certaine limite.

Ex 10. Nombres amicaux.

Deux nombres M et N sont appelés nombres_amis si la somme des diviseurs propres (sauf M) de M est égale à N et la somme des diviseurs propres de N est égale à M

Écrire une fonction amis qui retourne le nombre_amis (s'il existe) d'un nombre passé en paramètre , cette fonction utilise la fonction somme_div de l'exercice 8.

Écrire un programme principal qui affiche tous les nombres_amis inférieurs à une certaine limite.

Ex 11. Écrire une fonction « singleton » qui prend une liste en paramètres et renvoie « vrai » si la liste est égale à [0].

Ex 12. Écrire une fonction « même longueur » qui prend deux listes en arguments et renvoie vrai si les listes sont de même longueur.

Ex 13. Écrire une fonction « appartient » qui prend en entrée une liste et une variable du même type que les éléments de la liste, et renvoie « vrai » si l'élément appartient à la liste.

Ex 14. Écrire une fonction « bégaye » qui prend une liste en entrée, et renvoie en sortie une liste où tous les éléments sont doublés.

Ex 15. Écrire une fonction « croissante » qui admet comme paramètre une liste et retourne « vrai » si la liste est ordonnée croissante.

Ex 16.

1. Créer un tableau de 10 nombres aléatoires (ou plus) compris entre 1 et 10.
   Le trier ; on utilisera la méthode `valeurs_en_vrac.sort()` . Il est inutile de faire une affectation, recopiez juste cette instruction.
   À partir de ce tableau, en créer un autre qui contiendra les valeurs présentes dans le tableau trié, mais uniquement en un seul exemplaire.
   Il est très fortement conseillé d'utiliser la méthode `tableau_valeurs.append(bidule)` pour rajouter « `bidule` » en fin de tableau.
   On affichera les 3 tableaux
2. On reprend le programme précédent en le complétant. Le tableau initial contiendra 50 valeurs.
   Comme précédemment, on le trie puis on crée cette fois-ci deux tableaux :
   - Le tableau des valeurs présentes `tableau_valeurs` ;
   - Le tableau des effectifs correspondants à ces valeurs `tableau_effectifs` .

   On peut ensuite calculer la moyenne (remarque : on pouvait le faire dès le début, ce n'est pas le but de cet exercice !)

Ex 17. Ecrire un programme qui stocke la décomposition en facteurs premiers d'un nombre entier strictement positif dans un tableau et ensuite affiche les éléments de ce tableau sous la forme 18 = 2*3*3.

Ex 18. Ecrire un programme qui affecte des valeurs aléatoires comprises entre 13 et 50 (inclus) à un tableau de 10 entiers, trie le tableau par ordre croissant et l'affiche.

Ex 19. Ecrire une fonction qui retourne la factorielle d'un nombre (exemple $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1$).

Déduire de la solution précédente, une fonction qui permet le calcul du nombre de combinaisons de p éléments parmi q définie par : $\begin{pmatrix} n \\ p \end{pmatrix} = \frac{n!}{p!(n-p)!}$. Ensuite construire le triangle de Pascal en prenant en entrée le nombre de lignes à afficher.

Ex 20. Reproduction des lapins

Un couple de lapins a sa première portée à 2 mois, puis une portée tous les mois. Chaque portée est un couple de lapins. Tous les couples ainsi obtenus se reproduisent de la même manière.

1. On distinguera le nombre de couples de nouveaux lapins nouveaux, le nombre de couples de lapins ayant un mois, un_mois, et le nombre de couples de lapins « vieux » ayant 2 mois ou plus, vieux. Calculer « à la main » le nombre de couples de lapins de chaque type, ainsi que leur nombre total, pour les 10 premiers mois.
2. Écrire un algorithme nb_lapins calculant le nombre de lapins obtenus au bout de nb_mois mois à partir de nb_couples couples jeunes, et renvoyant le résultat.
3. Écrire un algorithme lapins_un_milliard calculant au bout de combien de temps les lapins sont plus d'un milliard (on supposera qu'aucun lapin ne meurt pendant cette période), en partant d'un couple de lapins.
4. Mettre en œuvre ces algorithmes dans un programme Python.

email prof : frederic.mandon@ac-montpellier.
Site prof : http://www.maths-info-lycee.fr/
Les devoirs à la maison peuvent être rendus à 2. Les intitulés d'exercices/devoirs rendus par email doivent impérativement contenir votre (vos) nom(s), ainsi que le nom de l'exercice. La date donnée par email n'est pas indicative, l'heure limite est minuit.

Utilisation des fichiers iPython notebook en cours.
Copier tous les fichiers en « .pnb » qui se trouvent dans « documents>devoirs>mandon », les mettre dans un dossier « documents>cours notebook ». Le but est que chacun travaille sur sa propre version du cours, et non sur celle du premier qui l'a modifié ! Les notebooks, ainsi que les cours et exercices, se trouvent aussi sur mon site.
Lancer ipython notebook avec le raccourci « notebook », ou bien dans le menu démarrer>anaconda>notebook. Cela peut prendre un certain temps. Le navigateur Firefox est lancé automatiquement.
Faire un upload du fichier que vous voulez ouvrir dans la fenêtre du navigateur (« Home », « Jupyter »), puis, une fois que le fichier est présent dans cette fenêtre, cliquer sur « upload » à nouveau.
Ensuite doucle-cliquer (ou simple cliquer, ça dépend des jours et des versions…) sur le fichier dans la liste, qui s'ouvre dans un nouvel onglet.
Un double clic sur une cellule vous permet de la modifier.
Ctrl-clic exécute la cellule (ça marche aussi si vous avez par erreur cliqué dans une cellule de texte)
Maj-clic exécute la cellule et passe à la suivante.
On peut effacer le résultat d'un programme avec un simple clic sur la cellule correspondante, puis menu cell > current output > clear.

Sous séquence 1 : introduction, variables, entrées-sorties (1 séance)
Algorithme d'Ahmès
TD1
Cours à lire/comprendre/assimiler I – II (langage naturel) – III – IV 1 et 2 – VI 1, 2 et 3.
A faire chez soi V – 1
A faire chez soi (sous IDLE) avec éventuellement début en cours : exercices 1 (conversion des degrés) et 2 (permutation circulaire de valeurs de variables)
Si vous faites chez vous l'exercice 2, envoyez-le-moi par email au plus tard le mardi soir précédent la deuxième séance.

Sous séquence 2 : itératives et conditionnelles (2 séances)
TD2 : boucles et tests
Cours à lire/comprendre/assimiler VI 4, 5 et 6.
A faire chez soi (sous IDLE) avec éventuellement début en cours : exercices 3 (résolution de l'équation du second degré), 4 (tables de multiplications), 5 (lignes d'étoiles)
TD3 : structurer un programme
Cours à lire/comprendre/assimiler VI 7 et 8.
A faire chez soi (sous IDLE) avec éventuellement début en cours : exercice 8 (jeu deviner un nombre), sous forme structurée. Si vous le faites chez vous, envoyez-le-moi par email au plus tard le mardi soir précédent la troisième ou quatrième séance, suivant votre avancement.
DM à rendre par email pour le 05/10: ex9 (nombres parfaits), commenté et structuré. Les volontaires peuvent faire en plus l'exercice 10 (nombres amicaux).
DM à rendre sur papier pour le 01/10: algorithme de l'exercice 9, sous la forme des algorithmes du TD2. Les volontaires peuvent faire en plus l'exercice 10.

Sous séquence 3 : listes (2 à 3 séances)
TD4 : les listes
Cours à lire/comprendre/assimiler VI 9 et 10
Cours à relire, commencer à assimiler le 3 : IV
Pour tous les exercices suivants, ainsi que le dm, les seules fonctions et méthodes autorisées sont `len(liste)` et `liste.append` ! Interdiction d'utiliser les autres méthodes données dans le cours, mais aussi ce que vous pourriez trouver sur le web (comme « count » qui est très pratique pour le dm...)
Exercices en cours et à la maison : exercices 11 à 15
Devoir à la maison: exercice 16 –ou autre que je vous donnerai– , programme par email et algorithme sur papier. Dates à préciser, au plus tard premiers jours des vacances de la Toussaint.