Programmation Python

CHAPITRE II

Modules

Indispensables

Younes Lakhrissi
Professeur à l'ENSA de Fès
Université Sidi Mohamed Ben Abdellah

Objectif du chapitre

- **Cours**
  - Module math
  - Module random
  - Module datetime
  - Module os
  - Module turtle

- **Travaux dirigés et pratiques**
  - **TP3** : plus de 10 exercices à résoudre et à développer

Les modules

Bibliothèques de Python

- Les fonctions intégrées automatiquement dans le langage sont les fonctions les plus utilisées, comme : len(), print(), input(), …
- Les autres sont regroupées dans des fichiers séparés que l'on appelle des modules
- **Les modules sont des fichiers qui regroupent des ensembles de fonctions**
- Nous allons explorer quelques uns :
  - 1 - Module math
  - 2 - Module random
  - 3 - Module datetime
  - 4 - Module turtle
  - 5 - Module os

Installation d'un module

- **pip**
  - Gestionnaire de packages et modules par défaut pour Python
- **Syntaxe**
  - Pour connaître la version installée : pip --version
  - Pour lister les packages installés : pip list
  - Pour installer un package : pip install nompackage
  - Pour désinstaller un package : pip uninstall nompackage
- **Siteweb de référence** de packages disponibles :
  - https://pypi.org/
  - *pypi : Python Package Index*

Module 1 : math

- Il fournit l'accès aux fonctions mathématiques définies par le standard C.
- Ces fonctions ne peuvent être utilisées avec les nombres complexes, utilisez les fonctions du même nom du module **cmath**

- **importer le module entier**
  - **import math**
  - print('racine carré de 16 : ', **math.sqrt(16)**)

- **importer que les fonctions utiles**
  - **from math import sqrt**
  - print('racine carré de 16 : ', **sqrt(16)**)

Module 1 : math

- **Constantes**
  - math.**pi** : La constante π = 3.141592…
  - math.**e** : La constante e = 2.718281…
  - math.**inf** : Un flottant positif infini. (infini négatif, -math.inf.) Équivalent au résultat de float('inf')
  - math.**nan** : Un flottant valant NaN. Équivalent au résultat de float('nan')

```
print('valeur de  pi  :', math.pi)   → valeur de  pi  : 3.141592653589793
print('valeur de  e   :', math.e)    → valeur de  e   : 2.718281828459045
print('valeur de  inf :', math.inf)  → valeur de  inf : inf
print('valeur de -inf :', -math.inf) → valeur de -inf : -inf
print('valeur de  nan :', -math.nan) → valeur de  nan : nan
```

Module 1 : math

- **Conversion angulaire**
  - **degrees(x)** : Convertit l'angle x de radians en degrés.
  - **radians(x)** : Convertit l'ange x de degrés en radians.
- **Fonctions trigonométriques**
  - **cos(x) :** Renvoie le cosinus de x radians.
  - **sin(x) :** Renvoie le sinus de*x* radians.
  - **tan(x) :** Renvoie la tangente de x radians.
  - **acos(x) :** Renvoie l'arc cosinus de x, en radians.
  - **asin(x) :** Renvoie l'arc sinus de x, en radians.
  - **atan(x) :** Renvoie l'arc tangente de x, en radians

Module 1 : math

- **Fonctions arithmétiques**
  - **factorial(x)** : Renvoie la factorielle de x.
  - **ceil(x)** : Renvoie la fonction plafond de x, le plus petit entier plus grand ou égal à x.
  - **floor(x)** : Renvoie le plancher de x, le plus grand entier plus petit ou égal à x.
  - **fabs(x)** : Renvoie la valeur absolue de x.
  - **fsum(c)** : Renvoie une somme flottante exacte des valeurs d'une collection.
  - **gcd(a, b)** : Renvoie le plus grand diviseur commun des entiers a et b.

Module 1 : math

- **Fonctions logarithme et exponentielle**
  - **pow(x, y)** : Renvoie x élevé à la puissance y.
  - **sqrt(x)** : Renvoie la racine carrée de x.
  - **exp(x)** : Renvoie e**x
  - **log(x, base=e) :** renvoie le logarithme naturel de x
    - Avec un argument : base = e.
    - Avec deux arguments, renvoie le logarithme de x dans la base donnée, calculé par log(x) / log(base).
  - **log10(x) :** Renvoie le logarithme de x en base 10. = log(x, 10).

Module 1 : math

- **Exemples à tester**
  - **from math import ***
  - print('valeur de PI : ', **pi**) ----------→ 3.14159
  - print('racine carré de 16 : ', **sqrt(16)**) ----------→ 4.0
  - print('cos(0) :', **cos(0)**) ----------→ 1.0
  - print('sin(pi) :', **sin(pi/2)**) ----------→ 1.0
  - print('fact(5) :', **factorial(5)**) ----------→ 120
  - print('plafond de 3.14 :', **ceil(3.14)**) ----------→ 4
  - print('plancher de 3.14 :', **floor(3.14)**) ----------→ 3
  - print('abs(-3.14):', **fabs(-3.14)**) ----------→ 3.14
  - print('pgcd de 10 et 25 :', **gcd(10,25)**) ----------→ 5
  - print(' 2^^10 :', **pow(2, 10)**) ----------→ 1024.0

Module 2 : random

- Python propose toute une série de fonctions permettant de générer des nombres aléatoires
- **Module random**
  - from **random** import ***
- **Génération aléatoire des réels :**
  - **random() :** crée des nombres réels **entre 0 et 1**
  - **uniform(a, b) :** crée des nombres réels **entre a et b**
- **Test :**

```
print ('1er  appel de random()  : ', random())          → 0.662548
print ('2eme appel de random()  : ', random())          → 0.720558
print ('1er  appel de uniform() : ', uniform(10,12.5)) → 10.94826
print ('2eme appel de uniform() : ', uniform(10,12.5)) → 11.64778
```

Module 2 : random

- **Génération aléatoire des entiers :**
  - **randrange(start, stop, step)** : génère un entier dans l'intervalle **[start , stop [** le pas entre les éléments est **step**
  - **randrange(start, stop)** : par defaut **step = 1**
  - **randrange(stop)** : par defaut **start = 0** et **step = 1**
  - **randint(a, b)** : retourne un entier dans l'intervalle **[a , b]**
- **Exemples**
  - randrange(6) : renvoie des valeurs entre 0 et 5
  - randrange(2 , 15) : renvoie des valeurs entre 2 et 14
  - randrange(2 , 15 , 4) : renvoie une des valeurs : 2, 6, 10, 14
  - randint(19, 20) : renvoie une des valeurs : 19, 20

exclu

inclus

```
print(randrange(6) )              →  3
print(randrange(2 , 15)  )        →  6
print(randrange(2 , 15 , 4))      →  14
print(randint(19 , 20) )          →  20
```

Module 2 : random

- **Fonctions pour les séquences:**
  - **choice(seq)** : retourne un élément aléatoire parmi les éléments de la séquence passée en paramètres.
  - **shuffle(seq)** : effectue un mélange de la séquence passée en paramètres. La fonction ne retourne rien.
  - **sample(seq, k)** : retourne une liste de k éléments choisis aléatoirement à partir de la séquence seq passée en paramètres.
- **Exemple :**

```
liste = [1, -5, -3, 5, 7 ]
print(choice(liste)) ------> -3
shuffle(liste)
print(liste)  ----------> [-5, 1, 7, 5, -3]
print(sample(liste, 3)) ---> [5, 7, -3]
```

Module 2 : random

- **Exercice 1 (corrigé)**
  - Affichez une liste de 20 nombres entiers aléatoires compris entre 50 et 100.
  - Contrôler la génération de ces nombres pour ne pas avoir des nombres dupliqués dans la liste
- **Exercice 2 (corrigé)**
  - Écrire un programme qui aide les enfants à apprendre les tables de multiplication.
  - Le programme propose 2 entiers au hasard et demande à l'utilisateur de fournir le produit.
  - 10 opérations seront proposées et la note finale sur 10 doit être affichée.

Module 3 : datetime

- Module qui contient les classes les plus utiles pour la manipulation du temps :
  - **Classe date**
    - Représente une date (année, mois et jour) dans le calendrier grégorien
  - **Classe time**
    - Représente le temps (h, m, s, …) dans une journée
  - **Classe datetime**
    - Représente dans le même objet l'information concernant la date (objet date) et le temps (objet time)
  - **Classe timedelta**
    - Représente une durée = différence entre deux dates ou heures

Module 3 : datetime
classe datetime

- **Module à importer**
  - from **datetime** import ***
- **Récupérer la date et l'heure actuelles**
  - a = datetime.now() ← **Locale**
  - b = datetime.utcnow() ← **UTC :** Coordinated Universal Time
- **Créer un objet datetime**
  - datetime(y, m, d, h, m, s, microsecond, tzinfo)

Arguments obligatoires

Arguments optionnels

- **Code de test**

```
a = datetime.now()                      a =  2022-10-02 12:30:14.775879
b = datetime.utcnow()                   b =  2022-10-02 11:30:14.775879
c = datetime(2022,11,18,3,15,30)        c =  2022-11-18 03:15:30
d = c.replace(2000,11,6)                d =  2000-11-06 03:15:30
```

Module 3 : datetime
classe date

- **Module à importer**
  - from **datetime** import ***
- **Récupérer la date d'aujourd'hui**
  - a = date.today()
- **Créer un objet date**
  - b = date(2022, 10, 3) ⟵ **(Année , Mois, Jour)**
- **Code de test**

```
a = date.today()                    a =  2022-10-02
b = date(2022,10,3)                 b =  2022-10-03
c = b.replace(2010,10,15)           c =  2010-10-15
```

Module 3 : datetime
classe time

- **Module à importer**
  - from **datetime** import ***
- **Récupérer l'heure actuelle**
  - **a** = datetime.now().time()
- **Créer un objet date**
  - time(h =0, m =0, s =0, microsecond=0, tzinfo=None)
  - b = time(14, 0, 30)

Tous les arguments sont par défaut

- **Code de test**

```
a = datetime.now().time()
b = time(14,0,30)
c = time()
d = c.replace(18,0,15)
```

```
a =  15:55:07.942317
b =  14:00:30
c =  00:00:00
d =  18:00:15
```

Module 3 : datetime
classe timedelta

- **Objectif :**
  - représente une durée = différence entre deux dates ou heures
- **Module à importer**
  - from **datetime** import ***
- **Créer un objet date**
  - datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)
- **Code de test**

```
d1 = timedelta(days=3,seconds=3600, minutes=60,hours=24,weeks=2 )
print(d1)

a = datetime(2022,12,18,3,15,30)
b = datetime(2022,11,18,1,15,30)
d2 = a - b
print(d2)
```

```
18 days, 2:00:00
30 days, 2:00:00
```

Module 3 : datetime

- **Attributs**

```
maDate = datetime(2022,11,18,15,30,00)
print("maDate  :", maDate)
print('Année   :', maDate.year)
print('Mois    :', maDate.month)
print('Jour    :', maDate.day)
print('Heure   :', maDate.hour)
print('Minute  :', maDate.minute)
print('Seconde :', maDate.second)
```

- **Résultat :**

```
maDate  : 2022-11-18 15:30:00
Année   : 2022
Mois    : 11
Jour    : 18
Heure   : 15
Minute  : 30
Seconde : 0
```

Module 3 : datetime

- **Méthode strftime()**
  - Convertit un objet en une chaîne de caractères selon un format donné
- **Exemples :**

```
maDate = datetime(2022,11,18,15,30,00)
print("maDate :", maDate)                          maDate : 2022-11-18 15:30:00

a = maDate.strftime("%A %d/%m/%Y, %H:%M:%S")          a : Friday 18/11/2022, 15:30:00
print("     a :", a)

b = maDate.strftime("%a %d %b %Y")                    b : Fri 18 Nov 2022
print("     b :", b)

c = maDate.strftime("%d %B %y")                       c : 18 November 22
print("     c :", c)

d = maDate.strftime("%j")                             d : 322
print("     d :", d)
```

Module 3 : datetime

Formatage d'une date

Format	Description	Exemple
%a	Jour de la semaine abrégé dans la langue locale.	Lu, Ma, ...
%A	Jour de la semaine complet dans la langue locale	Lundi, Mardi, …
%b	Nom du mois abrégé dans la langue locale.	janv., févr
%B	Nom complet du mois dans la langue locale.	janvier, février, ...
%y	Année sur deux chiffres (sans le siècle).	00, 01, ..., 99
%Y	Année complète sur quatre chiffres	2000, 2022, …
%H	Heure à deux chiffres	de 00 à 23
%M	Minutes sur deux chiffres.	00, 01, ..., 59
%S	Secondes sur deux chiffres.	00, 01, ..., 59
%c	Représentation locale de la date et de l'heure.	Lun. 3 octobre 2022 15:30:00
%x	Représentation locale de la date	3/10/2022
%X	Représentation locale de l'heure.	21:30:00
%j	Numéro du jour dans l'année sur trois chiffres.	002, 015, 366

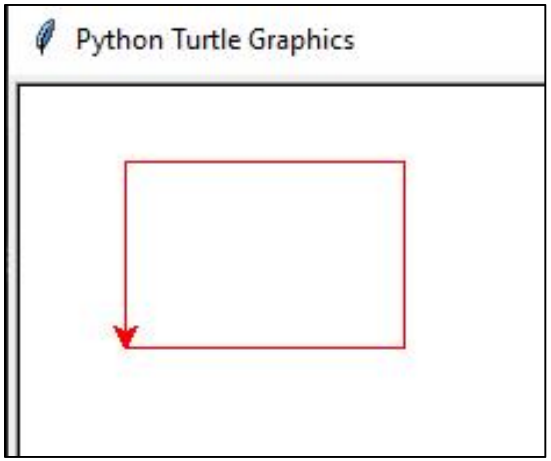
Module 3 : datetime

- **Exercice 3 :**
  - Ecrire un script qui demande à l'utilisateur de saisir sa date de naissance. Le programme affichera ensuite l'âge de l'utilisateur : le nombre d'années, de mois et de jours
- **Exercice 4 :**
  - Ecrire un script qui détermine si une année est bissextile ou non
  - une année bissextile est identifiée par les règles suivantes :
    - R1 : toute année divisible par 400
    - **OU** R2 : toute année divisible par 4 **ET** non divisible par 100
  - Exemples :
    - 2011 n'est pas bissextile. R1 (False) **OU** R2 (False) = False
    - 2008 est bissextile. R1 (False) **OU** R2 (True) = True
    - 1900 n'est pas bissextile, R1 (False) **OU** R2 (False) = False
    - 2000 est bissextile R1 (True) **OU** R2 (False) = True

Module 4 : turtle

- **Module à importer**
  - from **turtle** import ***
- **Objectif**
  - permet de réaliser des graphiques dits « tortue »,
  - dessins géométriques correspondant à la piste laissée par une tortue virtuelle, dont nous contrôlons les déplacements
- **Premier exemple :**
  - dessiner un rectangle rouge de dimension 120x80 pixels

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

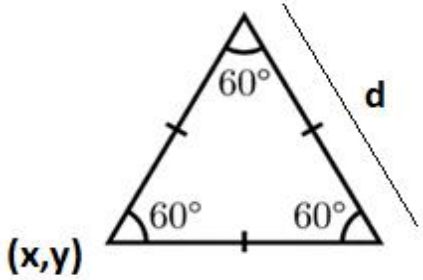


Module 4 : turtle

- reset() On efface tout et on recommence
- goto(x, y) Aller à l'endroit de coordonnées x, y
- forward(d) Avancer d'une distance donnée
- backward(d) Reculer
- up() Relever le crayon pour pouvoir avancer sans dessiner
- down() Abaisser le crayon (pour recommencer à dessiner)
- color(c) changer la couleur : 'red', 'blue', etc.
- left(angle) Tourner à gauche d'un angle donné en degrés
- right(angle) Tourner à droite d'un angle donné en degrés
- width(épai) Choisir l'épaisseur du tracé
- write(texte) Ecrire un texte (chaîne de caractères)
- setheading(angle) Oriente l'angle du dessin

Module 4 : turtle

- **Exercice 5** : Développez une fonction qui dessine un triangle équilatéral. Les variables x, y, et d doivent être fournies.



- **Solution**

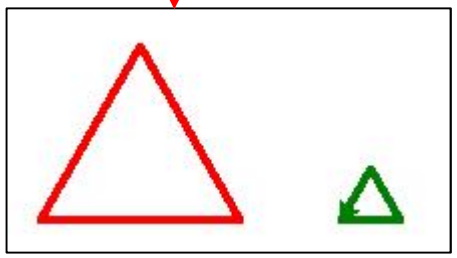
```
def dessinerTriangle(x, y, cote, couleur):
    color(couleur)
    pensize(4)
    up()
    goto(x,y)
    down()
    setheading(0)
    forward(cote)
    left(120)
    forward(cote)
    left(120)
    forward(cote)
```

```
clear()
dessinerTriangle(0, 0, 100, 'red')
dessinerTriangle(150, 0, 30, 'green')
```

1- Définition

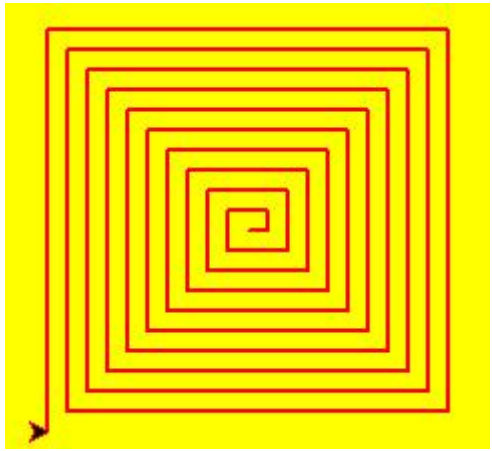
2 - Appel

3 - Résultat



Module 4 : turtle

- **Exercice 6** : Développez une fonction qui dessine une spirale carré qui ressemble à la figure suivante,
  Le nombre de tours est un paramètre à fournir à la fonction.

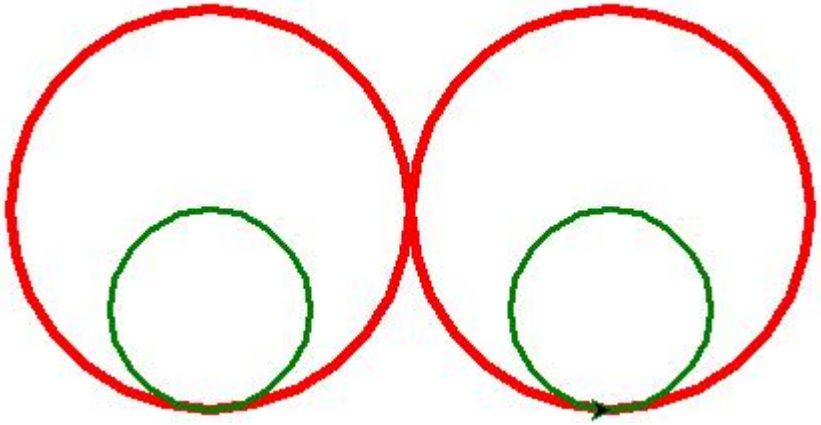


- **Solution**

```
def spiraleCarree(n): # Nombre de tours
    speed(10)
    pensize(2)
    pencolor("red")
    bgcolor("yellow")
    for i in range(1,2*n+1):
        forward(10*i)
        left(90)
        forward(10*i)
        left(90)

spiraleCarree(10)
```

Module 4 : turtle

- **Exercice 7** :
  - On peut déclarer un ou plusieurs objets explicitement
  - Créer deux objets x et y
  - x dessine les cercles rouges
  - y dessine les cercles verts



```
x = Turtle()
y = Turtle()
```

```
x.pencolor('red')
y.pencolor('green')

x.pensize(5)
y.pensize(3)

x.circle(100)
y.circle(50)

x.up()
x.goto(200,0)
x.down()

y.up()
y.goto(200,0)
y.down()

x.circle(100)
y.circle(50)
```

Module 5 : os

- **Manipulation des répertoires et fichiers**
  - from os import *
- **Le répertoire courant**
  - **getcwd**: get current working directory
  - repertoireActuel = **getcwd()**
  - print (’Le répertoire actuel est :’, repertoireActuel)
- **Contenu d’un répertoire**
  - **listdir()** : retourne la liste des sous-répertoires et les fichiers
  - mesRep = os.listdir()
  - print(mesRep)
- **Créer un répertoire**
  - **mkdir()** : make a new directory

Module 5 : os

- **Changer de répertoire**
  - **chdir**: Changing Directory
  - **chdir**('C:\Travail')
- **Renommer le nom d'un répertoire** ou fichier
  - os.**rename**('ancienRep', 'nouveauRep')
  - print(os.listdir())
- **Supprimer un répertoire vide**
  - os.**rmdir**('nouveauRep ')
  - print(os.listdir())
- **Supprimer un répertoire rempli**
  - **import shutil**
  - **shutil.rmtree**('repertoire')

Module 5 : os.path

- **os.path.isfile(path)**
  - Retourne True si le chemin path correspond à un fichier
- **os.path.isdir(path)**
  - Retourne True si le chemin path correspond à un répertoire
- **os.path.samefile(path1, path2)**
  - Retourne True si les chemins path1 et path2 font référence au même fichier ou répertoire. La comparaison est basée sur le i-node des documents,
- **os.path.exists(path)**
  - Retourne True si le chemin spécifié correspond à un répertoire ou à un fichier.

Module 5 : os

- **Exercice 8**
  - Calculer le nombre des sous-répertoires et le nombre des fichiers contenus dans le répertoire courant.

- **Solution**

```
contenuRep = os.listdir()
print('le nombre total :',len(contenuRep))
nbRep=0
nbFich=0
for x in contenuRep:
    if os.path.isdir(x):
        nbRep += 1
    else:
        nbFich += 1
print('dont', nbRep, 'sous-répertoires, et ', nbFich, 'fichiers')
```

Module 5 : os

- **Exercice 9**
  - Développer un script qui affiche le contenu d'un répertoire. Les sous-répertoires doivent être affichés en premier puis suivis des fichiers comme suivant :

```
Le répertoire XXX contient « nombre » documents :
-->  « Nombre des sous-répertoires » sous-répertoires :
     1- Rep1..
     2- Rep2..
     3- ..
-->  « Nombre des fichiers » fichiers :
     1- Fichier1..
     2- Fichier2..
     3- ..
```

Module 5 : os

- **Solution**

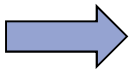
```
contenuRep = os.listdir()
print('le nombre total :',len(contenuRep))
listeSousRep = []
listeFichiers = []
for x in contenuRep:
    if os.path.isdir(x):
        listeSousRep.append(x)
    else:
        listeFichiers.append(x)

print('liste des fichiers :')
compteur = 1
for x in listeFichiers:
    print(compteur,x, sep=' ')
    compteur +=1

print('liste des sous repertoires :')
compteur = 1
for x in listeSousRep:
    print(compteur,x, sep=' ')
    compteur +=1
```

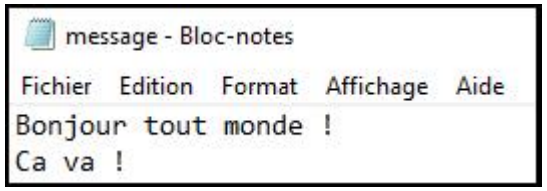
Gestion des fichiers

Texte et Binaires

Ecriture dans un fichier texte

- La fonction **open()** attend deux arguments :
  - Nom du fichier à ouvrir,
  - Mode d'ouverture.
- Modes d'ouverture :
  - 'w' (write) : un nouveau fichier vide est créé. Si le fichier existe déjà, celui-ci est effacé (écrasé).
  - 'a' (append)/ajout : les données sont ajoutées à la fin du fichier, à la suite de celles qui s'y trouvent déjà
- Code de test
  - monFichier = **open**('message.txt',**'w'**)
  - monFichier.**write**('Bonjour tout monde !\n')
  - monFichier.**write**('Ca va !')
  - monFichier.**close()**





Lecture d'un fichier texte

- Mode à utiliser pour la lecture :
  - 'r' (read) : le fichier doit exister déjà. si il n'existe pas, un message d'erreur va s'afficher
  - La fonction read() lit le contenu entier du fichier
- Code de test :
  - fichier = **open(**'message.txt', **'r')**
  - chaine = fichier.**read()**
  - fichier.close()
  - print('le contenu du fichier :')
  - print(chaine)

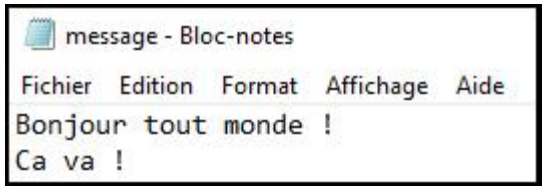
Lecture séquentielle d'un fichier

- La fonction read() peut posséder des arguments :
  - **read() :** retourne le contenu complet du fichier
  - **read(unEntier N)** : retourne N caractères à partir de la position actuelle atteinte.
- Code de test :

```
fichier = open('message.txt', 'r')
chaine1 = fichier.read(5)
chaine2 = fichier.read(5)
chaine3 = fichier.read(5)
fichier.close()
print('chaine 1:',chaine1)
print('chaine 2:',chaine2)
print('chaine 3:',chaine3)
```

- Résultat :

```
chaine 1: Bonjo
chaine 2: ur to
chaine 3: ut le
```



readline()

- Lecture d'un fichier ligne par ligne
- Le fichier 'message.txt' ne contient que deux lignes
- Code de test :

```
f = open('message.txt','r')
ligne1 = f.readline()
ligne2 = f.readline()
ligne3 = f.readline()
f.close()
print('--> lecture du fichier ligne par ligne:')
print('La 1ere ligne :',ligne1)
print('La 2eme ligne :',ligne2)
print('La 3eme ligne :',ligne3)
```

- Résultat:

```
--> lecture du fichier ligne par ligne:
La 1ere ligne : Bonjour tout le monde !

La 2eme ligne : Ca va !
La 3eme ligne :
```

Ligne vide

readline() et readlines()

- Lecture du fichier complet ligne par ligne :

```
f = open('message.txt','r')
while True:
    m = f.readline()
    if m == '':
        break
    print(m, end='')
f.close()
```

- readlines() : retourne une **liste** de chaine de caractères

```
f = open('message.txt','r')
liste = f.readlines()
f.close()
print('Contenu de la liste :',liste)
```

- Résultat:

```
Contenu de la liste : ['Bonjour tout monde !\n', 'Ca va !']
```

readline() et readlines()

- Notez bien que :
  - **readline()** renvoie une chaîne de caractères.
  - **readlines()** renvoie une liste.
- À la fin du fichier:
  - **readline()** renvoie une chaîne vide.
  - **readlines()** renvoie une liste vide.

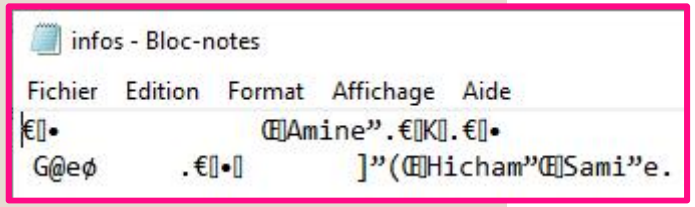
Ecriture/Lecture par variable

- read(), readline() et readlines() récupèrent les informations sous forme de chaines de caractères. **C'est problématique !**
- L'une des meilleures solutions consiste à importer un module Python spécialisé : le module **pickle**
- Le mode d'ouverture du fichier **doit être binaire** :
  - f = open('fichier', '**wb**')
  - f = open('fichier', '**rb**')
- Quelques fonctions :
  - **dump() :** attend deux arguments : 1) la variable à enregistrer, 2) l'objet fichier dans lequel on travaille.
  - **load() :** effectue le travail inverse, elle restitue chaque variable avec son type

Ecriture/Lecture par variable

```
import pickle
nom, age, taille, amis = 'Amine', 22, 175.75, ['Hicham','Sami']
#enregistrement des variables
f = open('infos.txt','wb')
pickle.dump(nom,f)
pickle.dump(age,f)
pickle.dump(taille,f)
pickle.dump(amis,f)
f.close()
#Lecture des variables
f = open('infos.txt','rb')
nom_r = pickle.load(f)
age_r = pickle.load(f)
taille_r = pickle.load(f)
amis_r = pickle.load(f)
f.close()
# affichage et vérification des information récupérées
print('affichage et vérification des informations récupérées')
print(nom_r, type(nom_r))
print(age_r, type(age_r))
print(taille_r, type(taille_r))
print(amis_r, type(amis_r))
```

Fichier binaire et non texte



- Résultat:

```
affichage et vérification des informations récupérées
Amine <class 'str'>
22 <class 'int'>
175.75 <class 'float'>
['Hicham', 'Sami'] <class 'list'>
```

Exercices

- **Exercice 10 :**
  - Développer une fonction qui permet de créer une copie du fichier texte passé en paramètre.
  - La copie doit être nommée avec le même nom de l'original mais terminé par '_bis'.
  - La première ligne dans la copie doit être :
    « cette copie est réalisée par moi-même : prénom + nom »

- **Exercice 11 :**
  - Ecrire une fonction qui compte le nombre de mots dans un fichier

Exercices

- **Exercice 12**
  - Ecrire une fonction qui crée un fichier nommé ‘bilan.txt’ qui trace les erreurs de ponctuation existantes dans le fichier passé en paramètres.
  - Règles de ponctuation à vérifier :
    - **R1** : La 1ere lettre d’une phrase doit être en majuscule
    - **R2** : Un paragraphe doit se terminer par un point « . »
    - **R3** : Un point « . » doit être précédé par une lettre et suivi par un espace ou un retour à la ligne
    - **R4** : Une virgule « , » doit être précédée par une lettre et suivie d’un espace
    - **R5** : Deux points « : » doit être précédé par un espace et suivi par un espace ou un retour à la ligne
  - Pour arriver à cet objectif, développez une fonction par règles.