

# Cours 5 : Approfondissements en python

- Notion de variables locales/globales
- Manipulation de fichiers textes

MPSI1 - Lycée Thiers

2014/2015

Variables locales, variables globales

Manipulation de Fichiers textes

# Variables globales, variables locales

# Variables locales, variables globales

- Une variable peut être globale ou locale selon qu'elle est définie :
  1. Dans l'environnement de l'appel d'une fonction : variable globale
  2. Dans une définition de fonction : variable locale

```
# Environnement du programme
x = 10                # x variable globale

# Définition d'une fonction fonct()
def fonct():
    y = 1             # z variable locale
```

- Une variable locale est créée à l'appel d'une fonction et détruite à la sortie de la fonction :

```
# Sortie de la fonction fonct()
print(y)
```

L'appel de la variable à l'extérieur de la fonction la définissant produit l'erreur :

```
NameError: name 'y' is not defined
```

# Variables locales, variables globales

- Une variable globale est accessible et modifiable dans l'environnement du programme où elle est définie :

```
# Environnement du programme
x = 10          # x variable globale

# Boucle while dans l'environnement
while (x > 0):
    x -= 1

# Définition d'une fonction fonct()
def fonct():
    print(x)     # on peut accéder à la valeur de x
    x = x + 1    # on ne peut pas modifier x
```

- Une variable globale est accessible par une fonction définie dans le même environnement, mais elle est non modifiable par cette fonction :

```
>>> 10          # Effet de print x
>>> x = x + 1    # Effet de la modification : erreur
UnboundLocalError: local variable 'x' referenced before assignment
```

# Variables locales, variables globales

- Si dans une fonction on crée une variable (locale) ayant même nom qu'une variable globale déjà définie, il s'agit d'une nouvelle variable qui est créée :

```
x = 10          # variable globale x
print(id(x))    # id() retourne l'identifiant de la variable x

def fonct():
    x = 1        # Variable locale créée
    print(id(x))
    x = x + 1    # Modifiable
    print(x)
```

Au lancement du programme :

```
4298180768 # Identifiant id(x) de x : variable globale
>>> fonct()    # Appel de la fonction
4298180984 # identifiant id(x) de x : variable locale de même nom
2         # La valeur de x retournée est celle de la variable locale
```

- Eviter de donner un même nom à une variable locale et une variable globale !

# Variables locales, variables globales

- Pour créer ou pour modifier une variable globale à l'intérieur d'une fonction, utiliser l'instruction `global` :

```
x = 10          # Variable globale : définie dans l'environnement

def fonct():
    global x     # Déclaration de la variable globale x
    x = x + 1    # Elle devient modifiable
    global y     # Déclaration d'une nouvelle variable globale y
    y = 2
```

```
>>> print(x)
10
>>> fonct()
>>> print(x)
11
>>> print(y)
2
```

- On évitera d'utiliser l'instruction `global` !

# Manipulation de Fichiers textes



# Manipulation de Fichiers textes

- En python, création et manipulation d'un fichier se font par l'intermédiaire d'un objet particulier, appelé objet-fichier, généré par la fonction :

`objet_fichier = open(nom du fichier, mode d'accès).`

Les paramètres sont des chaînes de caractère :

– `nom du fichier` est le nom du fichier, avec son extension.

– `mode d'accès` peut être :

1. `'w'` : (write) ouverture pour écriture seule. Lorsque le fichier n'existe pas il est créé dans le répertoire courant ; lorsque le fichier existe il est écrasé.
  2. `'a'` : (append) ouverture pour écriture seule. Lorsque le fichier n'existe pas il est créé dans le répertoire courant ; lorsque le fichier existe les données écrites le seront à la suite.
  3. `'r'` : (read) ouverture pour lecture seule. Le fichier doit exister dans le répertoire courant.
  4. `'+'` : ouverture pour lecture et écriture. Le fichier doit exister.
- Une fois la lecture et l'écriture dans le fichier terminés il faut refermer le fichier avec l'instruction : `objet_fichier.close()`

# Manipulation de Fichiers textes

Un objet-fichier admet les méthodes :

Lorsque ouvert en écriture :

- `write()` écrit une chaîne de caractère en fin de fichier ouvert en écriture.
- `writelines()` écrit une liste de chaînes de caractères en les concaténant.

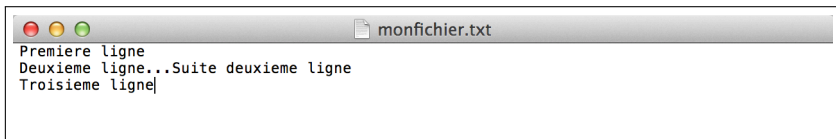
lorsque ouvert en lecture :

- `read()` lit l'intégralité du fichier.
- `readline()` lit la ligne suivante.
- `readlines()` retourne une liste contenant toutes les lignes du fichier.

On peut aussi parcourir le fichier ligne après ligne :

```
for ligne in objet_fichier
```

# Manipulation de Fichiers textes



```
monfichier.txt
Premiere ligne
Deuxieme ligne...Suite deuxieme ligne
Troisieme ligne
```

```
>>> objet = open('monfichier.txt','r')
>>> lect = objet.read()
>>> print(lect)
Premiere ligne
Deuxieme ligne...Suite deuxieme ligne
Troisieme ligne
>>> objet.close()
```

```
>>> objet = open('monfichier.txt','r')
>>> liste = objet.readlines()
>>> print liste
['Premiere ligne\n', 'Deuxieme ligne...Suite deuxieme ligne\n', 'Troisieme
ligne\n']
>>> objet.close()
```

# Manipulation de Fichiers textes

- Exemple : compter le nombre de ligne d'un fichier texte :

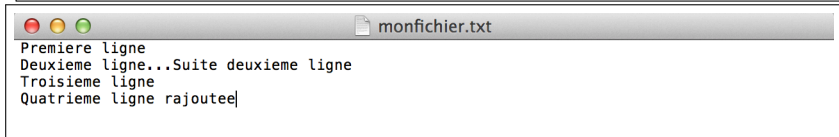
```
def nbreLignes(fichier):  
    objet = open(fichier,'r')  
    compteur = 0  
    for ligne in objet:    # Parcours des lignes du fichier  
        compteur += 1  
    return compteur
```

```
>>> nbreLignes('monfichier.txt')  
3
```

- Ajouter une ligne en fin d'un fichier texte :

```
def ajouteLigne(fichier,ligne):  
    objet = open(fichier,'a')  
    objet.write(ligne + '\n')  
    objet.close()
```

```
>>> ajouteLigne('monfichier.txt','Quatrieme ligne rajoutee')
```

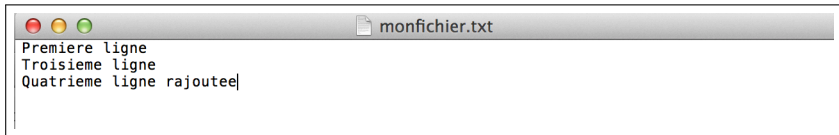


```
Premiere ligne  
Deuxieme ligne...Suite deuxieme ligne  
Troisieme ligne  
Quatrieme ligne rajoutee|
```

# Manipulation de Fichiers textes

- Exemple : supprimer une ligne d'un fichier texte :

```
def effaceLigne(fichier,i):  
    objet = open(fichier,'r')  
    liste = objet.readlines()  
    liste.pop(i-1)  
    objet.close()  
    objet = open(fichier,'w')  
    objet.writelines(liste)  
    objet.close()
```



# Manipulation de Fichiers textes

- Et pour stocker des données numériques ?

**Première méthode :** On peut le faire en les stockant sous forme de chaînes de caractères séparées par un séparateur; le retour à la ligne '\n' par exemple :

- Exemple : une fonction écrivant sous forme texte des données numériques issues d'une liste :

```
def wdata(fichier,liste):  
    f = open(fichier,'w')      # Ouverture en écriture  
    for x in liste:            # Parcours de la liste  
        # Ecriture de la donnée convertie en str + '\n'  
        f.write(str(x)+'\n')  
    objet.close()
```


- Lecture des données à partir du fichier et renvoi dans une liste :

```
def rdata(fichier):  
    f = open(fichier,'r')  
    liste = f.readlines()  
    f.close()  
    return [float(x) for x in liste]  
    # Convertir en float
```

# Manipulation de Fichiers textes

- Création d'un fichier datafile contenant les nombres entiers de 1 à 10.

```
>>> wdata('datafile',[1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```



```
1
2
3
4
5
6
7
8
9
10
```

- Lecture de ses données :

```
>>> print(rdata('datafile'))
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
```

- Ajout d'une donnée en fin de fichier :

```
def adata(fichier,x):    # Ajout d'un nombre en fin de fichier
    f = open(fichier,'a'); f.write(str(x)+'\n'); f.close()
```

```
>>> adata('datafile', 11)
```

# Manipulation de Fichiers textes

**Deuxieme methode** : On peut le faire en les stockant sur une ligne sous forme de chaînes de caractères séparées par un séparateur : un point-virgule ';' par exemple :

- Fonction écrivant des données numériques issues d'une liste :

```
def wdata(fichier,liste):  
    f = open(fichier,'w')      # Ouverture en écriture  
    for x in liste[:-1]:      # Parcours de la liste  
        # Ecriture de la donnée convertie en str + ';'   
        f.write(str(x) + ';')  
    f.write(str(liste[-1]))  
    objet.close()
```

- Lecture des données à partir du fichier et renvoi dans une liste :

```
def rdata(fichier):  
    f = open(fichier,'r')  
    ligne = f.readline()  
    liste = ligne.split(';')    # crée la liste des mots séparés par ';'   
    f.close()  
    result = []  
    for x in liste:  
        result.append(float(x))
```



# Manipulation de Fichiers textes

**Troisième méthode :** (la meilleure?) On stocke la liste des données numériques (convertie en chaîne de caractère).

- Fonction écrivant des données numériques issues d'une liste :

```
def wdata(fichier,liste):  
    f = open(fichier,'w')      # Ouverture en écriture  
    f.write(str(liste))  
    objet.close()
```

- Pour la lecture nous allons utiliser la fonction `eval()` : elle prend en paramètre une chaîne de caractère représentant une donnée, ou une instruction, et la convertit en ce qu'elle représente :

```
>>> eval('[1, 2]')  
[1,2]  
>>> eval('[1,2]\n')      # en ignorant les caractères spéciaux!  
[1, 2]  
>>> eval("print('bonjour')")    # Pour une instruction elle provoque l'exécution  
bonjour
```

```
def rdata(fichier):  
    f = open(fichier,'r')  
    ligne = f.readline()  
    liste = eval(ligne)      # eval() reconvertit en ce que ça représente  
    return liste
```