Partie III

Programmation fonctionnelle

Les fichiers

LES FICHIERS: *OPEN* ET *CLOSE*

Ouverture d'un fichier

```
objFichier = open(nom_fichier, mode_ouverture)
```

L'**ouverture** d'un fichier consiste en la création d'un **objet Python** de type fichier. C'est ensuite sur cet objet que sont faites les différentes actions (lecture, écriture, …). Le **nom du fichier** est une chaîne de caractères contenant l'extension du fichier (.txt, .dat, …). Par défaut le fichier se trouve dans le même répertoire que le programme, sinon il faut indiquer l'adresse complète du fichier. Le **mode d'ouverture** peut être **'r'** (lecture), **'w'** (écriture – crée un fichier ou écrase le fichier existant), **'a'** (écriture en fin de fichier), 'r+', 'w+' (lecture et écriture), 'a+' (lecture et écriture en fin de fichier).

Fermeture d'un fichier

```
objFichier.close()
```

Les fichiers: *OPEN* et *CLOSE*

```
# Ouvre le fichier "donnees.ini" en mode lecture
fsource = open("donnees.ini", 'r')

# Ouvre le fichier "donnees.txt", situé dans le dossier u:/Mes documents/Info/
# en mode écriture
fresu = open("u:/Mes documents/Info/donnees.txt", 'w')

# Demande le nom du fichier et l'ouvre en mode append.
filename = input("nom du fichier à afficher : ")
fsource = open(filename, 'a')

# Ferme l'objet fsource
fsource.close()
```

Les fichiers: Lecture et Ecriture

Lecture séquentielle des lignes du fichier

```
for ligne in objFichier:
```

Une ligne est une chaîne de caractère se terminant par le caractère de passage à la ligne '\n'.

Lecture d'une ligne d'un fichier

```
ligne = objFichier.readline()
```

Lecture d'un nombre de caractères consécutifs donné d'un fichier

```
texte = objFichier.read(taille)
```

La lecture d'un fichier se fait séquentiellement. Quand une ligne est lue, on ne peut revenir en arrière !

Ecriture dans un fichier

```
objFichier.write(texte)
```

On ne peut écrire que des chaînes de caractères.
Attention à ne pas oublier le passage à la ligne '\n ' si nécessaire.

Les fichiers: Lecture et Ecriture

```
In [5]: flecture = open("fable.txt",'r')

In [6]: for ligne in flecture:
   ...:     print(ligne[0],end = " ")
   ...:
L J E S L L C V P É C D L « A V M Q I V A L D J Q N L E À D C E L L M L L I L C L

In [7]: flecture.close()
```

```
In [13]: flecture = open("fable.txt",'r')

In [14]: ligne = flecture.readline()

In [15]: texte = flecture.read(4)

In [16]: print(ligne, texte)
L'AVANTAGE DE LA SCIENCE
 Jean

In [17]: flecture.close()
```

fable - Bloc-notes

Fichier Edition Format Affichage ?

```
L'AVANTAGE DE LA SCIENCE
Jean,de la Fontaine
Entre deux bourgeois d'une ville
S'émut jadis un différend :
L'un était pauvre, mais habile ;
L'autre, riche, mais ignorant.
Celui-ci sur son concurrent
Voulait emporter l'avantage ;
Prétendait que tout homme sage
Était tenu de l'honorer.
C'était tout homme sot ; car pourquoi révérer
Des biens dépourvus de mérite ?
La raison m'en semble petite.
« Mon ami, disait-il souvent
Au savant,
Vous vous croyez considérable ;
Mais, dites-moi, tenez-vous table ?
Que sert à vos pareils de lire incessamment ?
Ils sont toujours logés à la troisième chambre,
Vêtus au mois de juin comme au mois de décembre,
Ayant pour tout laquais leur ombre seulement.
La République a bien affaire
De gens qui ne dépensent rien !
Je ne sais d'homme nécessaire
Que celui dont le luxe épand beaucoup de bien.
Nous en usons, Dieu sait ! notre plaisir occupe
L'artisan, le vendeur, celui qui fait la jupe,
Et celle qui la porte, et vous, qui dédiez
À messieurs les gens de finance
De méchants livres bien payés. »
Ces mots remplis d'impertinence
Eurent le sort qu'ils méritaient.
L'homme lettré se tut, il avait trop à dire.
La guerre le vengea bien mieux qu'une satire.
Mars détruisit le lieu que nos gens habitaient :
L'un et l'autre quitta sa ville.
```

```
In [18]: fecriture = open("Nouveau_fichier.txt",'w')

In [19]: fecriture.write("Hello world")
Out[19]: 11

In [20]: fecriture.write("!")
Out[20]: 1

In [21]: fecriture.close()
```

Nouveau_fichier - Bloc-notes

Fichier Edition Format Affichage ?

```
Hello world!
```

Les fichiers: Manipulation des données

La lecture et l'écriture dans un fichier utilisent les **chaînes de caractères**.

Quelques méthodes utiles

Effacement du caractère à droite (par exemple \n)

```
ligne.rstrip()
```

```
In [22]: phrase = "0 1 2 \n"

In [23]: phrase.rstrip()
Out[23]: '0 1 2'
```

Sépare les éléments d'une chaîne (par défaut le séparateur est l'espace)

```
ligne.split()
```

```
ligne.split(sep)
```

```
In [24]: phrase.split()
Out[24]: ['0', '1', '2']

In [25]: phrase = "Hello_world"

In [26]: phrase.split("_")
Out[26]: ['Hello', 'world']
```

Partitionne une chaîne autour d'un séparateur. Renvoie la chaîne avant sep, sep, la chaîne après sep

```
ligne.partition(sep)
```

```
In [29]: phrase = "Nom: Calcium Ca"

In [30]: phrase.partition(":")
Out[30]: ('Nom', ':', ' Calcium Ca')
```

Partie IV

Programmation orientée objet

Objets et classes

Classes: Déclaration

```
class NomDeClasse:
    """Documentation"""
```

```
class NomDeClasse:
    """Documentation"""
    def __init__(self,parametres):
        instructions
```

```
nomObjet = NomDeClasse(parametres)
```

Appel du constructeur *__init__*

Classes: Les attributs

```
class NomDeClasse:
    """Documentation"""
    attributsDeClasse_ = valeur
    def __init__(self,parametres):
         self.attributs_ = valeur
        instructions
```

Tous les attributs doivent être initialisés dans le constructeur *__init__*

Passés en paramètre

```
def __init__(self, par1, par2, …):
        self.att1_ = par1
        self.att2_ = par2
```

ou Calculés en fonction des paramètres ou autres attributs

```
def __init__(self, par1, par2, …):
        self.att1_ = par1
        self.att2_ = par2
        self.att3_ = f(par1, par2 …)
```

ou Fixés

```
def __init__(self, par1, par2, …):
        self.att1_ = par1
        self.att2_ = par2
        self.att3_ = expression fonction de par1 et par2
        self.att4_ = constante
```

Classes: Les méthodes

```
class NomDeClasse:
    """Documentation"""
    attributsDeClasse_ = valeur
    def nom_methode(self,parametres):
         instructions
```

```
nomObjet.nom_methode(parametres)
```

```
def get_attribut(self):
```

```
def set_attribut(self,valeur):
```

Classes: Les méthodes spéciales

```
__init__
__add__
__mul__
__str__
__eq__
```

CLASSES: LE POLYMORPHISME

```
class NomDeClasse:
    """Documentation"""

    def nom_methode(self, parametres):
        instructions
```

```
class UneAutreClasse:
    """Documentation"""

    def nom_methode(self,parametres):
        instructions
```

CLASSES: LE POLYMORPHISME

```python
class Point:
    """Definit les points de l'espace 2D."""

    def __init__(self, x=0, y=0):
        " Constructeur avec valeurs par defaut des coordonnées "
        self.x_ = x
        self.y_ = y

    def deplace(self, dx, dy):
        """

        :param dx:  déplacement sur l'axe des x
        :param dy:  déplacement sur l'axe des y
        :return:    modifie les attributs de l
        """

        self.x_ += dx
        self.y_ += dy
```

```python
class Forme:
    """ Definit des formes geometriques ayant une couleur et un centre """

    def __init__(self, couleur, centre=None):...

    def deplace(self, dx, dy):
        """

        :param dx:  deplacement sur l'axe des x
        :param dy:  deplacement sur l'axe des y
        :return:    modifie la position du centre de la forme
        """
        # definition des nouvelles coordonnees du centre
        self.centre_.deplace(dx, dy)
```

Partie V

Programmation orientée objet

Sous-classes et héritage

Sous-classes: Déclaration

```
class NomDeSousClasse(NomDeClasse):
     """Documentation"""
```

```
class NomDeSousClasse(NomDeClasse):
    """Documentation"""
    def __init__(self,parametres):
         NomDeClasse.__init__(self,parametresClasse)
         instructions
```

```
nomObjet = NomDeSousClasse(parametres)
```

SOUS-CLASSES: L'HÉRITAGE

```
class NomDeClasse:
    """Documentation"""
    attributsDeClasse_ = valeur
    def nom_methode1(self, parametres1):
        instructions
```

```
class NomDeSousClasse(NomDeClasse):
    """Documentation"""
    def __init__(self, parametres):
        NomDeClasse.__init__(self,parametresClasse)
        instructions
    def nom_methode2(self, parametres2):
        self.nom_methode1(self, parametres1)
```

Sous-classes: Le polymorphisme

```python
class Fraction:
    """ modelise une fraction caractérisée par un numérateur et u
    """

    def __init__(self, num=0, den=1):...

    def __str__(self):...

    def normalise(self):...

    def __mul__(self, other):
        """..."""

        # on multiplie denominateurs et numérate
        num = self.num_ * other.num_
        den = self.den_ * other.den_

        # on crée une nouvelle fraction
        prod = Fraction(num, den)
        prod.normalise()

        return prod
```

```python
from fraction import Fraction

class Fraction_irreductible(Fraction):
    # une fraction irréductible est une Fraction
    # il y a héritage entre les deux classes

    def __init__(self, num=0, den=1):...

    def __add__(self, other):...

    def __mul__(self, other):
        fmul = Fraction.__mul__(self, other)
        fmul.set_irreductible()
        return fmul
```

Partie I

Structuration d'un programme

Compilateur, IDE…

- **Langage de programmation :**
  - notation conventionnelle
  - destinée à **formuler des algorithmes**
  - et produire des programmes informatiques qui les appliquent.
  - **moyen de communication** par lesquels le programmeur communique
    - **avec l'ordinateur**
    - mais aussi **avec d'autres programmeurs**.

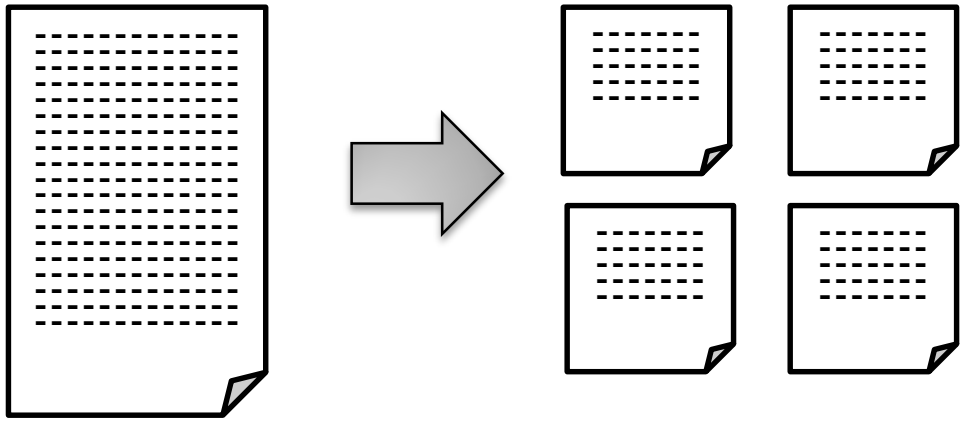
- **Interpréteur** :
  - outil ayant pour tâche d'**analyser, de traduire et d'exécuter les programmes** écrits dans un langage informatique.
  - langages dont les programmes sont généralement exécutés par un interpréteur qualifiés de de **langages interprétés**.

- **Environnement de développement** (IDE) :
  - Ensemble **d'outils pour augmenter la productivité des programmeurs**
    - **éditeur de texte** destiné à la programmation,
    - fonctions pour **démarrer l'interpréteur**
    - **débogueur en ligne** (exécuter ligne par ligne le programme en cours de construction.

Extraits de wikipedia

STRUCTURER UN PROGRAMME

Structure modulaire : Séparer un programme en plusieurs sous-parties



- Plus simple à **comprendre**
- Plus simple à **corriger / modifier**
- Permet de **séparer le travail** (entre les membres d'une équipe, sur différentes périodes de temps…)
- Permet de faire des 'sous-validations'

Le programme principal (ou main)

- ✓ Lancer les éléments du programme
- ✓ Tester le programme (penser aux différentes configurations)

```
if __name__ == « __main__ »:
```

Bonnes Pratiques

Guide de bonnes pratique de Guido von Rossum (créateur de Python) :
https://www.python.org/dev/peps/pep-0008/ (en anglais)
https://larlet.fr/david/biologeek/archives/20080511-bonnes-pratiques-et-astuces-python/

- Aérer le texte
  - Entourer les fonctions, classes par 2 lignes blanches
  - Entourer les méthodes d'une ligne blanche
  - Ajouter des lignes blanches pour séparer des sections logiques de code
  - Ajoutez un espace après ", " et après ":" dans les dictionnaires mais pas avant.
  - Mettez des espaces autour des assignements et des comparaisons
  - Pas d'espace aux ouvertures/fermetures de parenthèses ou juste avant une liste d'arguments.
  - Pas d'espace en ouverture/fermeture de docstrings
- Eviter les lignes trop longues
  - Essayer de limiter les lignes à environ 79 caractères

Bonnes Pratiques

- Nommage
  - Noms de fonctions au format: lowercase_with_underscores
  - Noms de classes au format: CapWords
  - Noms de paramètres: format mixedWords, lowercase ou lowercase_with_underscores
  - Noms de modules: format lowercase ou lowercase_with_underscores
- Commentaires
  - Eviter les commentaires évidents (`x = x + 1 # Incrémente x`)
  - Eviter les commentaires qui contredisent le code
- Documentation
  - Ecrire des docstring pour tous les modules, fonctions, classes et méthodes

TESTER UN PROGRAMME

- Identifier les cas d'étude

Exemple: racine(s) réelle(s) d'un polynôme de degré 2

$$ax^2 + bx + c = 0$$

$b^2 - 4ac > 0$	$b^2 - 4ac = 0$	$b^2 - 4ac < 0$
2 racines	1 racine	0 racine

- Tester chaque fonction, chaque module

Exemple: racine(s) réelle(s) d'un polynôme de degré 2

```
racine (a,b,c)
```

`racine (1,1,0)`	`racine (1,-2,1)`	`racine (1,0,1)`

Déboguer un programme

- Lire une erreur et la comprendre
- Tester et corriger
- Débogueurs intégrés