

L3 Informatique - 2021-2022

Atelier 4 – Chaînes de caractères



Objectifs de ce cours

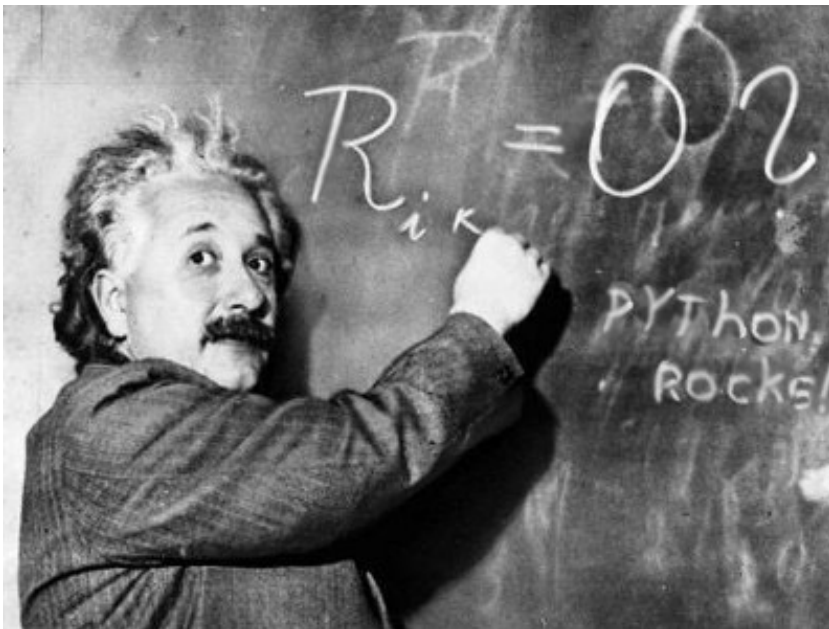


- Vous donner les connaissances minimales pour réaliser les exercices de l'atelier 4
- Maîtriser la particularité des chaînes de caractères en python
- Vous initiez à l'utilisation d'expressions régulières pour effectuer des recherches

Plan du cours

Chaînes de caractères

- Les chaînes de caractères en python
- Les expressions régulières
- Les fichiers de texte



POWERFUL AND
OPEN TOOLS FOR
SCIENTISTS.



Chaînes de caractères en python

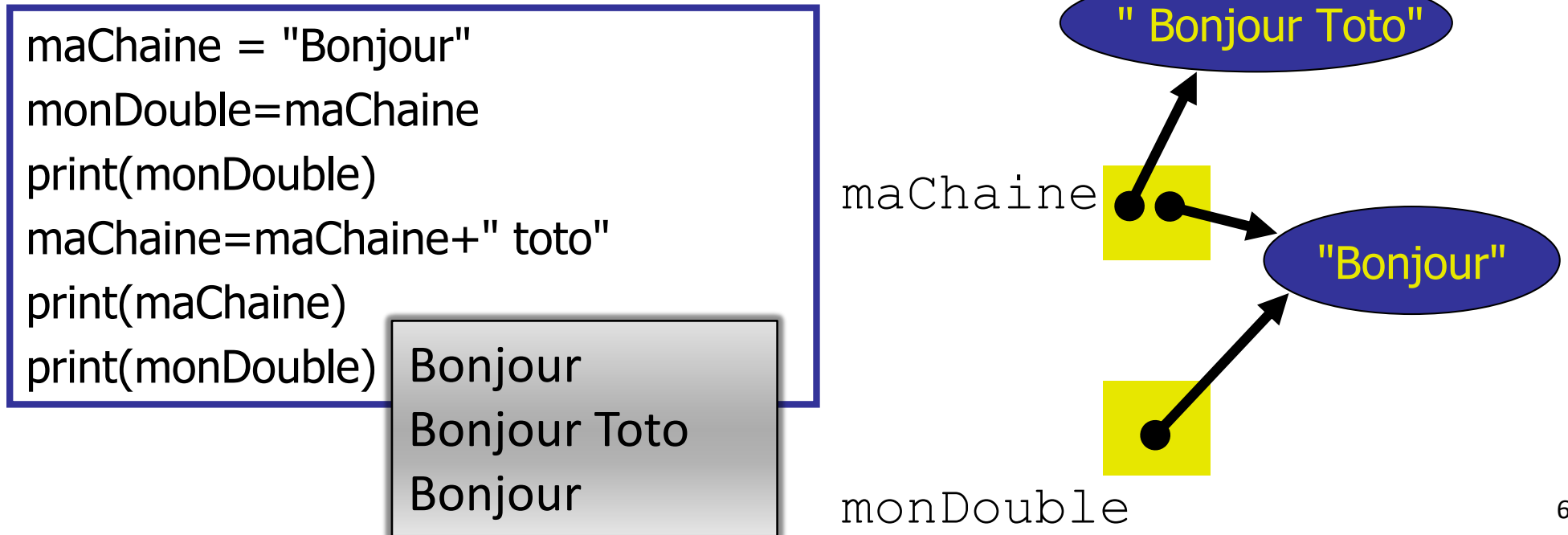
Première entrée dans le
mode objet de python

Les chaînes de caractères

- Objets de la **classe** str (ou string)
- Objets **immutables**
 - non modifiables
 - Listes particulières car non modifiables
- Objets indexables
- Objets itérables

Notion d'objet immutable

- Les chaînes de caractères sont des objets qui ne peuvent pas être modifiés, ils sont dits « **immutables** »
- Si l'on tente de les modifier une **nouvelle instance est créée.**



Méthodes de la classe str

Méthodes *m* applicables sur une chaîne *c* : *c.m()*

- **lower()** - **upper()**

- **renvoie** le texte transformé en minuscules (resp. en majuscules)

- **capitalize()**



```
t="bonjour"  
print(t[0].upper()+t[1:])
```

- retourne met la chaîne avec la première lettre en majuscule

- **endswith()** (*suffixe*[, *debut*[, *fin*]])

- retourne True si la chaîne se termine par *suffixe*, et False sinon
- *suffixe* peut aussi être un tuple de suffixes
- le test peut débuter à *debut* et se terminer à *fin*

Exemple de Manipulation d'une chaîne

```
def remplacer_car(ch:str,num:int,c:char)->str:
```

```
#renvoie la chaîne ch dans la quelle le caractère d'indice num est  
remplacé par le caractère c
```

```
    ch[num-1]=c  
    return ch
```

*Ch[num-1]=c
TypeError: 'str' object does not support item assignment*

```
#Test
```

```
t= "bonjour"
```

```
r=remplacer_car(t,4,"p")
```

```
print(r)
```



Attention !!

Ne pas oublier que les chaînes sont immutables
ch ne peut pas être modifiée !!

Exemple de Manipulation d'une chaîne

```
def remplacer_car(ch:str,num:int,c:char)->str:  
#renvoie la chaîne ch dans la quelle le caractère d'indice num (en  
démarrant à 1) est remplacé par le caractère c  
    ch_res=ch[0:num-1]+c+ch[num:]  
    return ch_res
```

#Test

```
t= "bonjour"  
r=remplacer_car(T,4,"p")  
print(R)
```

bonpour

Exemples de Parcours d'une chaine

```
def affiche_car(ch:str) :  
    #affiche les caractères de la chaine ch de manière verticale  
    for c in ch :  
        print(c)
```

```
def o_majuscules(ch:str) :  
    #renvoie la chaine Ch en remplaçant les « o » par des « O »  
    ch_res=""  
    for c in ch :  
        if c=='o':  
            ch_res+= c.upper()  
        else:  
            ch_res+= c  
    return ch_res
```

#Test

```
t= "bonjour"  
affiche_car(t)  
r=o_majuscules(t)  
print(r)
```

```
b  
o  
n  
j  
o  
u  
r  
bOnjOur
```

Méthodes

Attention ! la chaîne d'origine n'est pas modifiée !!

- **replace**(c1:str, c2:str)->str:
 - renvoie la chaîne d'origine dans laquelle la sous-chaîne c1 a été remplacée par la chaîne c2 dans la chaîne
- **count**(c:str)->int :
 - renvoie le nombre d'occurrences de la sous-chaîne c dans la chaîne

Méthodes de test de caractères

- **isalpha()**-> booléen:
 - renvoie True si la chaine n'est composée que de caractères alphabétiques (lettres)
- **isdecimal()**-> booléen:
 - renvoie True si la chaine n'est composée que de caractères décimaux(chiffres)
 - **isdigit()** et **isnumeric()** fonctionnent de manière similaire mais incluent également les nombres exprimés avec des caractères spéciaux
 - ex: '\u00BD' index caractère unicode ½

Méthodes de test de caractères

- `isspace()`->booléen:
 - renvoie True si la chaîne n'est composée que d'espaces
- `strip([chars:str])`-> str:
 - renvoie une chaîne dans laquelle les caractères apparaissant dans `chars` sont supprimés
 - si `chars` est omis, les espaces sont supprimés

Méthodes

- **split**([séparateur]) : liste de chaines
 - renvoie une liste de mots extraits de la chaine

```
animaux = "girafe:tigre:singe"  
print(animaux.split(":") )  
# affiche ['girafe', 'tigre', 'singe']
```

séparateur par défaut:
espace ou tabulation

- **expandtab**([nbCar]): chaine
 - renvoie une copie de la chaine dans laquelle les caractères de tabulation (\t) ont été remplacés par des espaces en fonction du nombre de caractères par colonne (nbCar=8 par défaut)

Méthodes

attention L ne doit
contenir que des
chaînes de caractères

- **join**(lst:list)->str:
 - transforme une liste (un itérable) en une chaîne de caractère en ajoutant le séparateur spécifié entre chaque mot de la liste

chaineSeparateur.join(l)

```
l=['girafe', 'tigre', 'singe']
```

```
res="-".join(l)
```

```
#renvoie la chaine 'girafe-tigre-singe'
```

```
res=" ".join(l)
```

```
#renvoie la chaine 'girafe tigre singe'
```



Expressions régulières

Expressions régulières



Existent dans la plupart des langages de programmation

Expressions régulières

- Une expression régulière (**regex**) est une chaîne de caractère qui spécifie le format d'un ensemble de chaînes de caractères
- Elle utilise des symboles qui ont une signification particulière:
 - . ^ \$ * + ? { } [] \ | ()

Interêts
Recherches et
Modifications concises

Regexp	chaines correspondantes
corte	« corte » « universite de corte » « ville de corte » « cortenais »
cort*	« corte » « universite de corte » « ville de corte » « cortenais » « corps »
^cort+e	« corte » « cortenais » « cortte »

Caractères spéciaux

Caractère	Signification
.	n'importe quel caractère sauf retour chariot
['a','b', 'c']	un caractère appartenant à l'énumération
[^'a', 'b']	Tous les caractères sauf ceux énumérés
^	début de la chaine, la ligne...
\$	fin d'une chaine, ligne...
 (a b)	alternative - ou reconnaît l'un ou l'autre
(...)	désigne un groupe de caractères pouvant se répéter
*	0, 1 ou plusieurs occurrences du caractère qu'il suit
+	1 ou plusieurs occurrences du caractère qu'il suit
?	0 ou 1 occurrence du caractère qui suit

Répétitions de caractères

suite de caractères



Sequence{ nb1,nb2}

- $A\{2\}$: la lettre A (en majuscule) se répète 2 fois consécutives.
- $BA\{1,9\}$: la séquence BA se répète de 1 à 9 fois consécutives.
- $BRA\{,10\}$: BRA n'est pas présent du tout ou présent jusqu'à 10 fois consécutives.
- $VO\{1,\}$: VO est présent au moins une fois.

Un petit exercice

GR(A)?S	GRAS	TRUE	FALSE	TRUE
GR(A)?S	GRS	TRUE	FALSE	TRUE
M(.)+N	MAISON	TRUE	FALSE	TRUE
M(.)+(O)+N	MAISON	TRUE	FALSE	TRUE
M(.)+([a-z])+N	MAISON	TRUE	FALSE	FALSE
M(.)+([A-Z])+N	MAISON	TRUE	FALSE	TRUE
^!	!MAISON!	TRUE	FALSE	TRUE
!MAISON	!MAISON!	TRUE	FALSE	TRUE
^!MAISO!\$!MAISON!	TRUE	FALSE	FALSE
^!MAISON!\$!MAISON!	TRUE	FALSE	TRUE
^!M(.)+!\$!MAISON!	TRUE	FALSE	TRUE
([0-9])	03 88 00 00 00	TRUE	FALSE	TRUE
^0[0-9]([.-/?[0-9]{2}){4}	03 88 00 00 00	TRUE	FALSE	TRUE
^0[0-9]([.-/?[0-9]{2}){4}	03/88/00/00/00	TRUE	FALSE	TRUE
^0[0-9]([.-/?[0-9]{2}){4}	03_88_00_00_00	TRUE	FALSE	FALSE

Caractères et séquences d'échappement

- **Caractère d'échappement**
 - un caractère placé devant un caractère spécial (ou une séquence) dans le but d'éviter son interprétation
- **Séquence d'échappement**
 - séquence constituée d'un caractère d'échappement suivi des caractères qu'il modifie

Caractères et séquences d'échappement

- `\` est un caractère d'échappement
 - `\\` représente le caractère `'\'`
 - `\\n` représente la séquence de caractères `'\'` and `'n'`
- En python, pour plus de visibilité, on utilise le caractère `r` placé avant une expression spéciale
 - `r"\n"` représente la séquence de caractères `'\'` and `'n'`

Séquences d'échappement spéciales

- `\d` : uniquement des chiffres (équivalent à `[0-9]`)
- `\D` : aucun chiffre (équivalent à `[^0-9]`).
- `\s` : un espace (équivalent à `[\t\n\r\f\v]`).
- `\S` : aucun d'espace (équivalent à `[^\t\n\r\f\v]`).
- `\w` : uniquement des caractères alphanumériques (équivalent à `[a-zA-Z0-9_]`).
- `\W` : aucun caractère alphanumérique (équivalent à `[^a-zA-Z0-9_]`).

Expressions régulières (en python)

- Bibliothèque python **re**
- **Fonctions:** findall, search, match, sub
- Classe **Regex:**
 - Objet expression régulières
 - Méthodes: findall, search, match, sub
- Classe **match Object:**
 - objet résultat de l'application d'une regex sur une chaîne
 - Méthodes: group, groups

Fonctions de recherche

- **findall**(regex, chaine) : liste de chaines
 - renvoie une liste contenant toutes les sous chaines qui vérifient l'expression régulière
 - None si regex n'est pas trouve

```
print(re.findall("[0-9]+", "Bonjour 111 Aurevoir 222"))  
#affiche ['111', '222']
```

Fonctions de recherche

- **split**(regex, chaine) : liste de chaines
 - renvoie une liste contenant toutes les sous chaines délimitées par les occurrences de la regex
 - None si regex n'est pas trouve

séparateurs



```
print(re.split("[\s,;]+", "Bonjour ; Aurevoir ; adieu"))  
# affiche ['Bonjour', 'Aurevoir', 'adieu']
```

Fonctions de recherche

Objet de la
classe_
sre.SRE_Match

- **search**(regex, chaine) : Match Object
 - renvoie un match object correspondant à l'application de la regex sur la chaine
 - None si aucune correspondance n'est trouvée
 - **match**(regex, chaine) : idem mais la recherche ne s'effectue qu'à partir du début de la chaine

```
chaine = ""  
expression = r"^0[0-9]([ .-]?[0-9]{2}){4}$"  
while re.search(expression, chaine) is None:  
    chaine = input("Saisissez un numéro de téléphone (valide) :")
```

Test de la validité
d'un numéro de
téléphone

Fonction de remplacement

- **sub**(regex, rempl,*chaine*, nb) : chaine
 - renvoie une chaine dans laquelle nb occurrences des sous-chaines de *chaine* qui vérifient regex sont remplacées par la chaine rempl
 - si regex comporte des groupes: renvoie une liste de tuples
 - si nb=0 ou est omis: toutes les occurrences sont remplacées

```
c="Bonjour * Vous êtes très gentil *"  
print(re.sub(r"(\*)","Monsieur Dupont",c))  
# affiche  
# Bonjour Monsieur Dupont Vous êtes très gentil Monsieur Dupont
```

Groupes

- Permettent de décomposer une expression
- Utiles pour les remplacements
- Les groupes sont numérotés à partir de 1

Groupe 1 → (a)b(cd) ← Groupe 2

- On peut également les nommer: (?P<nomGroupe>...)

Bonjour (?P<nom>\w+) de (?P<adresse>\w+)

Utilisation des groupes

```
phrase="Bonjour Marie de Corte"
res=re.search(r"Bonjour (?P<nom>\w+) de
(?P<adresse>\w+)", phrase)
print(res.group('nom') + " -" + res.group('adresse'))
#affiche Marie - Corte
```

méthode renvoyant
une chaîne



```
texte= "Jean;Corte;Marie;Ajaccio;Paul;Bastia"
print(re.sub(r";?(?P<nom>\w+);(?P<adresse>\w+)",
            r"Bonjour \g<nom> de \g<adresse>\n", texte))
```

#affiche

#Bonjour Jean de Corte

#Bonjour Marie de Ajaccio

#Bonjour Paul de Bastia

- ;? : commence par 0 ou 1 ;
- (?P<nom>\w+); groupe *nom* de 1 ou plusieurs caractères alphanumériques
- (?P<adresse>\w+) ; groupe *adresse* de 1 ou plusieurs caractères alphanumériques

Regex compilées

- Il est possible de compiler une regex pour ensuite l'appliquer sur plusieurs chaînes
- **compile**(regex) : object regex
 - renvoie un objet regex associé à la *regex* transmise en paramètre

```
chn_mdp = r"^[A-Za-z0-9]{6,}$"  
exp_mdp = re.compile(chn_mdp)  
mot_de_passe = ""  
while exp_mdp.search(mot_de_passe) is None:  
    mot_de_passe = input("Tapez votre mot de passe : ")
```



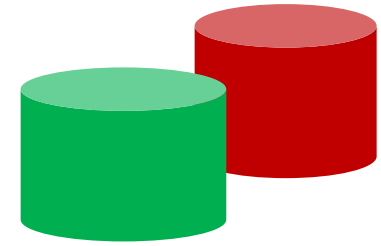

Fichiers en python

Généralités

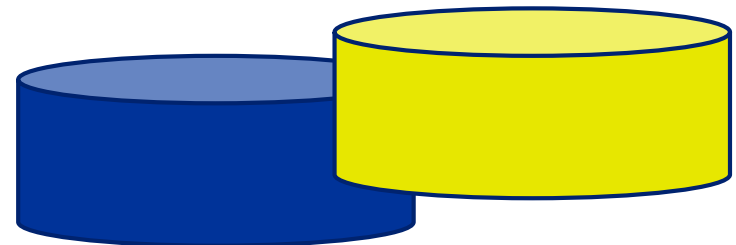
Fichiers binaires

Fichiers de texte

Notion de fichier



- Rôle fondamental en informatique: besoin de persistance des données
- Premier moyen de stockage « logique » des données
- Unique solution avant l'apparition des bases de données
- Mais les bases de données utilisent des fichiers!!



Notion de fichier: deux niveaux de définition

■ Niveau physique (système d'exploitation):

- Suite d'octets stockée sur un support physique permanent et repérée par un nom

```
00100010 00101100 00111100 00101101  
00100010 00101100 00111100 00101101  
01100010 00111100 00111000 00101101  
10100010 10101100 11111100 00101111
```

■ Niveau logique (applications):

- Suite de données mémorisée de manière persistante
- Différents types de fichiers selon :
 - la nature des données stockées
 - les manipulations envisagées



Types de fichiers

- Au niveau physique, tous les fichiers sont des suites d'octets.
- Au niveau logique, on distingue deux types selon la manipulation :
 - **Fichiers « binaires »**: suites d'octets indifférenciés



- **Fichiers « de texte »**: suites de caractères lisibles dans un éditeur de texte

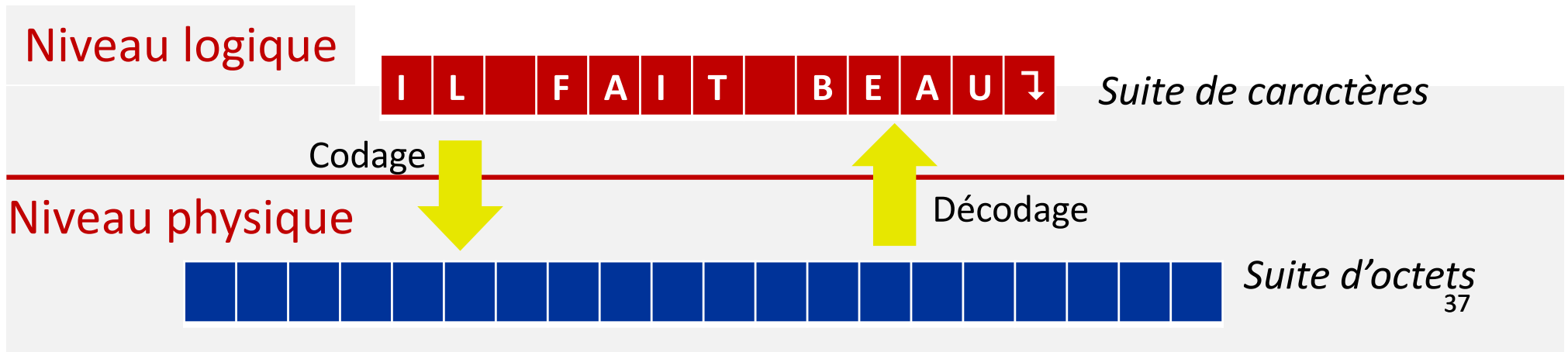
encodage/décodage
automatique



Fichiers de texte

- Suites de caractères
- Associés à un standard d' **encodage** de caractères : tables associant chaque caractère à un ou plusieurs octets
- Exemple
 - codage UNICODE : 1 caractère = 2 octets

fichiers dits « ASCII »



Types d'encodages de caractères

Pour plus de détails

<http://www.alsacreations.com/astuce/lire/83-codages-ascii-latin1-etc.html>

- ASCII : standard américain

- 128 caractères (pas de lettres accentuées ...)

- ISO-8859-1 ou Latin-1

- Extension de l'ASCII (191 caractères dont les accents !)
- ISO-8859-15 ou Latin-9 (extension avec œ et €)

- Codage cp1252

- Extension de ISO-8850-1 (218 caractères)

Codage windows
par défaut

- Codage UTF-8

- le plus général (famille du standard Unicode)
- codage sur 1 à 4 octets
- utilisé par 87.6% en 2016 des sites web

Codage Unix -
Linux par défaut

Types de fichiers de texte

- Fichiers de texte simples :
 - Suite de caractères organisés en lignes (repérées par un caractère « retour chariot »)
- Fichiers de textes structurés avec balises
 - Chaque donnée élémentaire est encadrée par un balisage
 - fichiers HTML, XML, Latex, etc.
- Fichiers d'enregistrements
 - Contient des enregistrements (valeurs de type structurée)
 - ex= nom, age, adresse

Exemples de fichiers d'enregistrements

Paoli;Pasquale;0495156487;ubabbu@yahoo.fr
Bonaparte;Napoleon;0156912347;empereur@free.fr
LeGrand;Alexandre;0289765194;alegrand@free.fr
DeGaulle;Charles;0149875231;cdg@aol.fr

enregistrements en lignes
séparateur ;

- Occupation optimisée de l'espace
- Gestion complexe

Paoli	Pasquale	0495156487	ubabbu@yahoo.fr
Bonaparte	Napoleon	0156912347	empereur@free.fr
LeGrand	Alexandre	0289765194	alegrand@free.fr
DeGaulle	Charles	0149875231	cdg@aol.fr

enregistrements en lignes
champs de largeur fixe

- Perte de place
- Traitements facilités

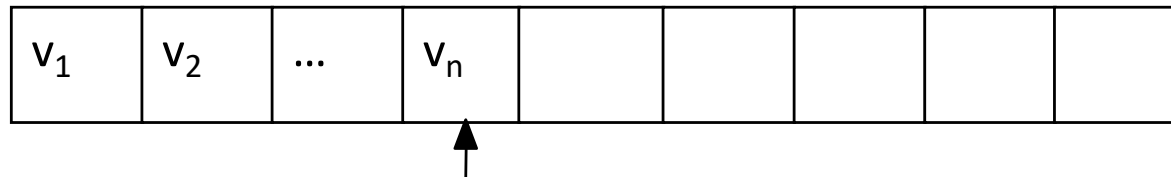
Fichiers binaires

- Désigne tout fichier contenant autre chose que des caractères:
 - fichier image (bitmap)
 - fichier de son
 - fichier pdf
 - ...
- Données non lisibles directement
- Nécessite un programme spécifique pour être lu et exploité.
- Plus compact et plus rapide à lire/écrire

Modes d'accès aux données

■ Accès séquentiel :

- On accède au contenu dans l'ordre du stockage
 - Pour lire la valeur v_n il faut avoir lu les valeurs v_1 à v_{n-1}



valeur=octet ou
caractère

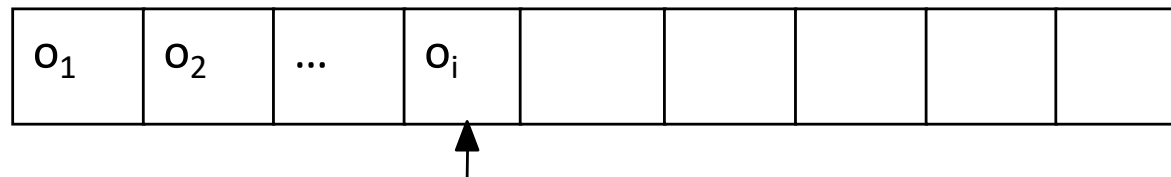
Possible pour tout type de fichier

Modes d'accès aux données

- **Accès direct :**

accès dit
« aléatoire »
(random)

- On accède directement au i ème octet du fichier

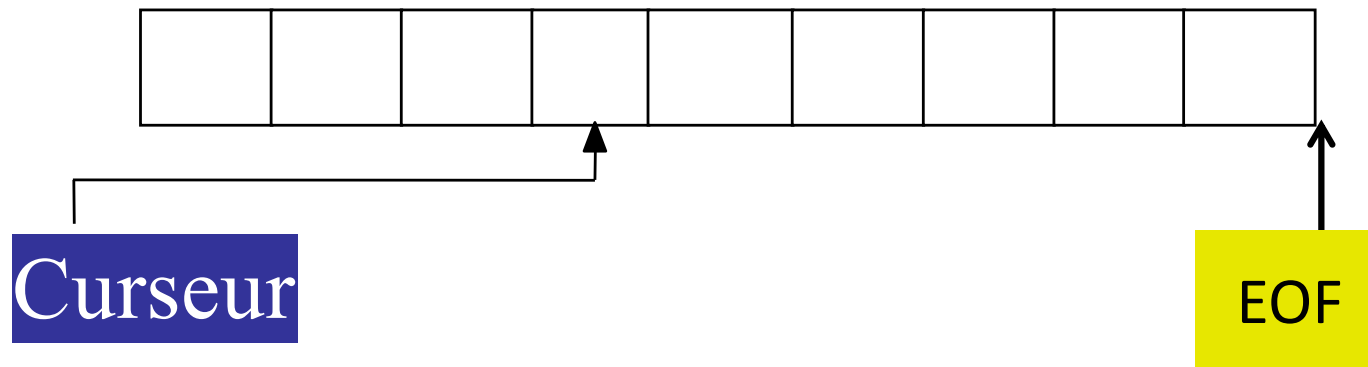


- Toujours possible pour les fichiers binaires :

- Accès direct au i ème octet

Principes de manipulation d'un fichier

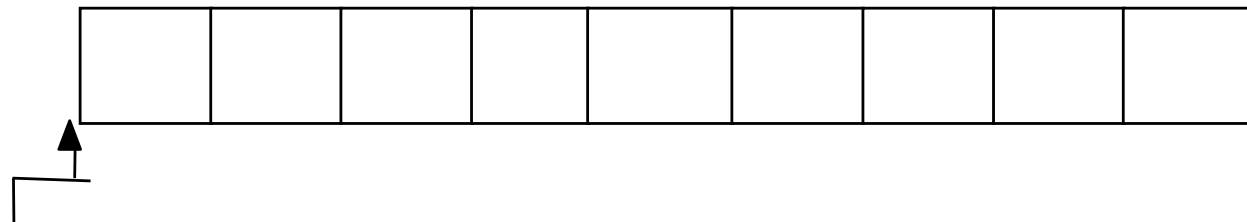
- Un indicateur de position (**curseur** de parcours) indique la **position courante**.



- Une position particulière: fin de fichier
 - **EOF** : End of File
 - En général, une fonction booléenne permet de savoir si la position EOF est atteinte par le curseur

Etapes de manipulations d'un fichier

1. Ouverture du fichier



Indicateur de position

2. Lectures/Ecritures dans le fichier

3. Fermeture du fichier

Fichiers en python

- Un fichier en python est un objet de la classe Fichier:
 - Manipulation par les méthodes de la classe Fichier
- La distinction fichier de texte/fichier binaire est définie lors de l'ouverture du fichier:
 - paramètre mode de la méthode open
 - t (texte) mode par défaut
 - b(binaire) à préciser de manière explicite
- La manipulation d'un fichier en python passe par les étapes classiques.

Ouverture d'un fichier

- **open**(nomfic, mode, encoding) : objet fichier
 - Renvoie un **objet fichier** associé au fichier nomfic
 - **mode** d'ouverture
 - r : lecture seule (valeur par défaut)
 - w : écriture seule (écrasement du fichier existant de même nom; si le fichier n'existe pas, il est créé)
 - a: ajout en fin de fichier (si le fichier n'existe pas , il est créé)
 - x: crée un nouveau fichier et l'ouvre en écriture
 - r+, w+, a+ : lecture et écriture (mode modification)
 - **encoding**: par défaut cp1252 sous windows et UTF8 sous Unix/Linux
- Après ouverture, le curseur est en position 0
- uniquement pour les fichiers de texte

Ouverture d'un fichier

- Par défaut, le fichier est ouvert en mode texte (t)
- Il faut ajouter **b** aux différents modes pour l'ouvrir en mode binaire (lecture/écriture d'octets et non de caractères)
 - **rb**: ouverture en mode lecture binaire
 - **wb**: ouverture en mode écriture binaire
 - **ab**: ouverture en mode ajout binaire
 - **rb+**, **wb+** ou **ab +**: lecture et écriture binaire

Particularité des fichiers binaires

- Intérêt:
 - lecture **octet par octet** ou blocs d'octets.
 - accès direct possible.
- Opérations de lectures/écriture nécessitant un **decodage/encodage explicite**.
- Utilisation des méthodes read/write avec des paramètres ou résultats de type tableau d'octets (**bytes**)

Ouverture d'un fichier

- Après l'exécution de la méthode open:
 - mode r,w : curseur positionné à 0
 - mode a : curseur positionné à la fin du fichier

Fichier python

curseur



Si le curseur est en position i ,
une opération de lecture lira l'octet $i+1$

Encodage/Decodage de caractères

- méthode **encode**(*chaine*, *typEncodage*):
tableau d'octets
 - renvoie la liste d'octets correspondant à la conversion de la chaine de caractère *chaine*
- méthode **decode**(*tabOct*, *typEncodage*):*chaine*
 - renvoie la chaine de caractères correspondant à la conversion du tableau d'octets *tabOct*

cf.fichiers binaires

Autres fonctions utiles:

- **ord**(*ca*) renvoie le code du caractère *ca*
- **chr**(*co*) renvoie le caractère de code *co*

Lecture d'un fichier

Le curseur est
déplacé de
nbCar/nbOctets

- **read**([nbCar/nbOctets]) : chaîne
 - Fichier de **texte**: renvoie **une chaîne de caractères** correspondant à nbCar caractères lus dans le fichier à partir de la position du curseur
 - Fichier **binaire**: Renvoie un tableau d'octets de nbOctets lus à partir de la position du curseur
 - si nbCar est omis ou <0: **renvoie l'intégralité du fichier**

Lecture d'un fichier de texte

- **readLine()** : chaine
 - Renvoie une seule ligne du fichier à partir de la position du curseur
 - la ligne se termine par le caractère \n sauf si c'est la dernière
 - **Renvoie la chaine vide si fin de fichier**
- **readLines()** : liste de chaines
 - Renvoie toutes les lignes du fichier à partir de la position du curseur dans une liste avec une ligne par élément

Manipulation du curseur de position

- **tell()** : entier Renvoie 0 quand le fichier vient d'être ouvert

- Renvoie la position courante du curseur dans le fichier mesurée en nombre d'octets depuis le début du fichier

- **seek(nbOctets, pos)**

attention : le nombre d'octets n'est pas toujours égal au nombre de caractères

- déplace le curseur du fichier de nb octets
- à partir de :
 - du début si pos=0
 - de la position courante si pos=1
 - de la fin du fichier si pos=2

Intéressant pour les fichiers binaires

Écriture dans un fichier

- **write**(texte/tabOct): entier
 - écriture de la chaîne de caractères *texte* (ou du tableau *tabOct*) dans le fichier à partir de la position du curseur
 - Si ouverture en mode w: écriture en écrasant le contenu existant
 - Si ouverture en mode a : écriture à la fin du fichier
 - renvoie le nombre de caractères (octets) écrits

Pour définir des lignes, il faut explicitement insérer les caractères de saut de ligne \n

Fermeture d'un fichier

- `close()`
 - ferme le fichier et libère toutes les ressources utilisées
- `closed`
 - attribut indiquant si le fichier est ouvert (True) ou fermé(False)

Instruction with

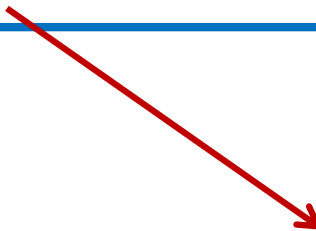
- Permet une fermeture propre des ressources
 - L'appel à close() est fait automatiquement

```
with open("profs.txt" , r) as f :  
    print(f.read())
```

Lecture globale d'un fichier de texte (avec read)

```
# ouverture du fichier en lecture (r=read)
f=open("profs.txt","r")
tout=f.read() #variable de type chaine de caractères
print("** Contenu du fichier **")
print(tout)
print("** fin **")
```

tout est de type
string



```
** Contenu du fichier **
Simonnet
Nivet
Vittori
Poggi
Cagnard
** fin **
```

Lecture globale d'un fichier de texte (avec readlines)

```
# ouverture du fichier en lecture
```

```
f=open("profs.txt","r")
```

```
listeTout=f.readlines()
```

```
print("** Contenu du fichier **")
```

```
print(listeTout)
```

```
print("** fin **")
```



```
** Contenu du fichier **
```

```
['Simonnet\n', 'Nivet\n', 'Vittori\n', 'Poggi\n', 'Cagnard']
```

```
** fin **
```

listeTout est de type
liste de string

Lecture ligne par ligne avec readline

ouverture du fichier en lecture

```
f=open("profs.txt","r")
```

```
c=f.readline()
```

```
print("** Contenu du fichier **")
```

```
while c!="":
```

```
    print(c)
```

```
    c=f.readline()
```

```
print("** fin **")
```

```
** Contenu du fichier **
```

```
Simonnet
```

```
Nivet
```

```
Vittori
```

```
Poggi
```

```
Cagnard
```

```
** fin **
```

#autre écriture plus concise

```
for ligne in f:
```


```
    print(ligne)
```

Les \n sont conservés
et donc affichés d'où
les lignes vides

Fichier d'enregistrements à longueur variable

- Exemple de fichier

Utilisation de
séparateurs (ici ;)



```
Simonnet;0610142522  
Nivet;0652555555  
Vittori;0688888888  
Poggi;0614555555  
Cagnard;0689999999
```

Accès séquentiel à un enregistrement

```
def lireProf(f, indice):  
    #renvoie une chaine de caractère représentant le prof numéro "indice"  
    dans le fichier f  
    #l'enregistrement est supposé présent  
    f=open(f,"r") # ouverture du fichier en lecture  
    #lecture de indice-1 enregistrements  
    for i in range(1,indice):  
        f.readline()  
    #lecture de l'enregistrement indice  
    prof=f.readline()  
    #création d'une liste avec une valeur par champ  
    lprof=prof.rstrip().split(";")  
    return lprof
```

"Nivet;0652555555/n"

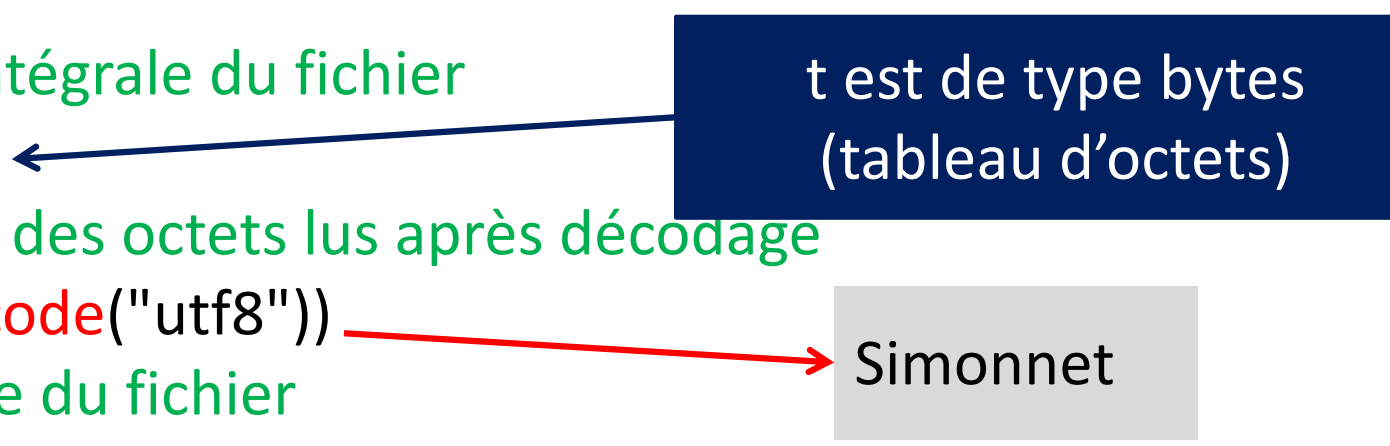
['Nivet', '0652555555']

`rstrip([chars])` : élimine les caractères situés en fin de ligne (par défaut \n)

`split([séparateurs])`: renvoie une liste de chaines selon les séparateurs

Manipulation binaire d'un fichier

```
# ouverture du fichier profs en mode lecture/ecriture binaire)
f = open('profs','bw+')
# écriture de la chaine « Simonnet » après encodage
f.write("Simonnet".encode("utf8"))
#positionnement au début du fichier
f.seek(0)
#lecture intégrale du fichier
t=f.read ()
#affichage des octets lus après décodage
print(t.decode("utf8"))
#fermeture du fichier
f.close()
```



Le module struct contient des fonctions facilitant le codage/décodage d'un fichier binaire