Séance 3 : Boucles for et chaînes de caractères

L1 – Université Côte d'Azur

Création du répertoire de travail

Avant de commencer ce TP : sur votre ordinateur, si ce n'a pas été déjà fait lors des deux premiers TP, créez un répertoire `Python` dans lequel vous rangerez tous vos fichiers de l'UE. Dans votre répertoire `Python` créer un répertoire `TP3`. Durant cette séance, tous vos fichiers devront être sauvegardés dans ce répertoire `TP3`.

Les prochaines semaines, vous devrez créer les répertoires `TP4`, puis `TP5`. Ainsi il sera facile de vous retrouver dans tous les fichiers.

Exercice 1 — Échauffement (⋆)

1. Faites afficher le code ASCII des caractères `'A'`, `'D'`, `'0'` (zéro), `'3'` puis vérifiez sur la table ASCII page suivante ; voyez-vous comment on peut déduire le code ASCII de toute lettre majuscule à partir de celui de `'A'`, et de tout chiffre à partir de celui de `'0'` ?
2. Faites calculer à Python la position dans l'alphabet de la lettre M (réponse : 12 si on compte à partir de 0, sinon 13 pour les humains) ;
3. Faites afficher à Python les caractères de code ASCII compris entre 32 et 44 sur une même ligne. Le résultat obtenu doit être le suivant où ␣ représente le caractère espace :

   ␣!"#$%&'()*+,
4. Définissez une fonction `est_chiffre(c)` qui renvoie `True` si le caractère `c` est un chiffre. Vous n'utiliserez pas la méthode `isdigit`, ce serait de la triche. Bon, ou alors uniquement une fois que vous avez vu comment faire sans. Écrire des tests avec `assert` pour votre fonction.
5. Définissez une fonction `masquer_numéro(s)` qui renvoie la chaîne `s` dans laquelle les chiffres sont remplacés par des étoiles. Testez la dans le shell. On n'utilisera pas la fonction `print` mais seulement un `return`.

```
>>> masquer_numéro('Bonjour je vends 1 chat. Appellez au 0678912345.')
'Bonjour je vends * chat. Appellez au **********.'
```

Exercice 2 — Cryptographie antique (⋆⋆)

Un système cryptographique ancien, souvent appelé code de César, consiste à choisir une clé entière `k` entre 1 et 25 pour fabriquer, à partir d'un message `msg`, un nouveau message codé avec la technique suivante. Chaque lettre majuscule de `msg` est décalée de `k` positions vers la droite (l'alphabet est circulaire : après `'Z'` on revient sur `'A'`). Les autres caractères du message sont laissés intacts !

1. Programmez la fonction `code_césar_lettre(c,k)` qui retourne le caractère correspondant au chiffrement du caractère contenu dans la variable `c`. Écrivez des tests avec `assert`.

```
>>> code_césar_lettre('A', 3)
'D'
>>> code_césar_lettre('Y', 3)
'B'
```

```
>>> code_césar_lettre(' ', 3)
' '
>>> code_césar_lettre('a', 3)
'a'
```

2. Programmez la fonction `code_césar(msg,k)` qui retourne le message codé, en appliquant la transformation précédente à chaque caractère.

```
>>> code_césar('LES GAUGAU... LES GAUGAU... LES GAULOIS !!!', 10)
'VOC QKEQKE... VOC QKEQKE... VOC QKEVYSC !!!'
```

3. Sur le même modèle, programmez la fonction `décode_césar(msg,k)` qui prend un message codé et retourne le message en clair.
4. Défi urgent : décodez le message `'WZG GCBH TCIG QSG FCAOWBG'` dont César a perdu la clé !

Exercice 3 — Table ASCII (⋆⋆)

Écrivez un programme qui affiche la table ASCII ci-dessous en respectant la mise en page.

```
 32       33 !     34 "     35 #     36 $     37 %     38 &     39 '
 40 (     41 )     42 *     43 +     44 ,     45 -     46 .     47 /
 48 0     49 1     50 2     51 3     52 4     53 5     54 6     55 7
 56 8     57 9     58 :     59 ;     60 <     61 =     62 >     63 ?
 64 @     65 A     66 B     67 C     68 D     69 E     70 F     71 G
 72 H     73 I     74 J     75 K     76 L     77 M     78 N     79 O
 80 P     81 Q     82 R     83 S     84 T     85 U     86 V     87 W
 88 X     89 Y     90 Z     91 [     92 \     93 ]     94 ^     95 _
 96 `     97 a     98 b     99 c    100 d    101 e    102 f    103 g
104 h    105 i    106 j    107 k    108 l    109 m    110 n    111 o
112 p    113 q    114 r    115 s    116 t    117 u    118 v    119 w
120 x    121 y    122 z    123 {    124 |    125 }    126 ~
```

```
for i in range(32,127):
    if i<100:
        print(' ',end='')
    print(i,chr(i),sep=' ',end='  ')
    if (i+1)%8 == 0: # retour à la ligne tout les 8 symboles
        print('')
print()
```

Exercice 4 — Print Proust (⋆ ⋆ ⋆)

Écrivez une fonction `justification_à_gauche(s,n)` qui prend une chaîne de caractères `s` contenant une phrase (sans symbole `'\n'`) et qui l'affiche à l'écran en mettant au plus `n` caractères par ligne, en justifiant à gauche.

```
>>> justification_à_gauche("Mais au lieu de la simplicité, c'est le faste que…" , 12)
Mais au lieu
de la
simplicité,
c'est le
faste que…
```

Testez votre programme en affichant cette phrase [1] de Proust en entier [2] sur 80 caractères par ligne.

Mais au lieu de la simplicité, c'est le faste que je mettais au plus haut rang, si, après que j'avais forcé Françoise, qui n'en pouvait plus et disait que les jambes « lui rentraient », à faire les cent pas pendant une heure, je voyais enfin, débouchant de l'allée qui vient de la Porte Dauphine – image pour moi d'un prestige royal, d'une arrivée souveraine telle qu'aucune reine véritable n'a pu m'en donner l'impression dans la suite, parce que j'avais de leur pouvoir une notion moins vague et plus expérimentale – emportée par le vol de deux chevaux ardents, minces et contournés comme on en voit dans les dessins de Constantin Guys, portant établi sur son siège un énorme cocher fourré comme un cosaque, à côté d'un petit groom rappelant

1. oui, c'est une seule phrase !
2. Si nécessaire le texte est disponible à l'adresse : https://upinfo.univ-cotedazur.fr/~obaldellon/L1/py/tp3/marcel-proust.html

le « tigre » de « feu Baudenord », je voyais – ou plutôt je sentais imprimer sa forme dans mon coeur par une nette et épuisante blessure – une incomparable victoria, à dessein un peu haute et laissant passer à travers son luxe « dernier cri » des allusions aux formes anciennes, au fond de laquelle reposait avec abandon Mme Swann, ses cheveux maintenant blonds avec une seule mèche grise ceints d'un mince bandeau de fleurs, le plus souvent des violettes, d'où descendaient de longs voiles, à la main une ombrelle mauve, aux lèvres un sourire ambigu où je ne voyais que la bienveillance d'une Majesté et où il y avait surtout la provocation de la cocotte, et qu'elle inclinait avec douceur sur les personnes qui la saluaient.

```python
def justification_à_gauche(s,n) :
    mot = ''
    ligne = ''
    taille_ligne = 0
    taille_mot = 0
    for i in range(len(s)) :
        if s[i] != ' ' : # Si on est à l'intérieur d'un mot
            mot = mot + s[i]
            taille_mot = taille_mot + 1
        else : # Si on a fini de lire un mot
            if taille_ligne+taille_mot+1 <= n and taille_ligne > 0 :
                # Si le mot rentre dans la ligne
                ligne = ligne + ' ' + mot
                taille_ligne = taille_ligne + taille_mot + 1
            else :
                # Sinon, on commence une nouvelle ligne
                if taille_ligne > 0 :
                    print(ligne)
                ligne = mot
                taille_ligne = taille_mot
            # On remet mot et taille_mot à zéro
            mot = ''
            taille_mot = 0
    # Il reste à s'occuper du dernier mot
    if taille_ligne + taille_mot != 0 :
        if taille_ligne + taille_mot > n :
            print(ligne)
            print(mot)
        else :
            print(ligne,mot)
```

Exercice 5 — Attaque sur un code de sécurité sociale (⋆⋆-⋆ ⋆ ⋆)

On se propose d'étudier une méthode pour chiffrer et déchiffrer un message m à l'aide d'une clé k. Le message m et la clé k sont des chaînes de caractères qui ne comportent que les caractères `'0'` et `'1'`. L'opération à la base du procédé est l'opérateur *ou exclusif*, noté $\oplus$, et défini comme suit : si c_1 et c_2 sont des caractères distincts, $c_1 \oplus c_2$ est le caractère `'1'`, sinon c'est le caractère `'0'`.

1. Écrivez une fonction `xor(c1,c2)` qui prend en arguments deux caractères c_1 et c_2 et qui renvoie le caractère correspondant au *ou exclusif*[3] $c_1 \oplus c_2$. Par exemple, `xor('0','1') == '1'`.

```python
def chr_xor(c1, c2) :
    return '0' if c1 == c2 else '1'
```

2. Le chiffrement de m avec la clé k est la chaîne de caractères e de la même longueur que m dont le i-ème caractère est le résultat du *ou exclusif* entre le i-ème caractère de m et le i-ème caractère de k ; si m est plus long que k, on répète la clé k à la suite d'elle-même pour obtenir une chaîne de caractères de la même longueur que m. Par exemple, si la clé est `'01'` et que m comporte 5 caractères, on rallonge k en `'01010'`.

3. En anglais et un informatique l'opérateur *ou exclusif* se note `xor`

Écrivez une fonction `chiffrement(m,k)` qui renvoie le chiffrement de m avec la clé k. Par exemple, on aura :

`chiffrement('1110011','10') == '0100110'`.

```
def chiffrement(m,k) :
    res = ''
    for i in range(len(m)) :
        c1=m[i]
        c2=k[i % len(k)]
        res = res + chr_xor(c1,c2)
    return res
```

3. On veut utiliser ce schéma de chiffrement pour coder un numéro de sécurité sociale comportant 15 chiffres décimaux. Pour cela, il suffit de convertir le numéro de sécurité sociale en une chaîne de caractères 0 ou 1, puis d'appliquer la fonction `chiffrement`. Écrivez une fonction `binaire(ss_id)` qui fait cette première étape : l'argument `ss_id` est une chaîne de caractères contenant uniquement des chiffres de 0 à 9, et la fonction renvoie la suite des écritures de ces chiffres en base 2, chacun sur 4 bits. Par exemple, `binaire('0123')` renvoie
`'0000000100100011' == '0000' + '0001' + '0010' + '0011'`

```
def binaire_chiffre(n):
    res = ''
    k = 8 # k va prendre les valeurs 8, 4, 2 et 1 (puissances de deux décroissantes)
    for i in range(4) :
        if n >= k :
            res = res + '1'
            n = n - k
        else :
            res = res + '0'
        k = k//2
    return res
```

Prenez l'habitude de tester vos fonctions.

```
>>> for i in range(10):
...     print(i,":",binaire_chiffre(i))
0 : 0000
1 : 0001
2 : 0010
3 : 0011
4 : 0100
5 : 0101
6 : 0110
7 : 0111
8 : 1000
9 : 1001
```

```
def binaire(s):
    res = ''
    for c in s:
        res = res + binaire_chiffre(int(c))
    return res
```

4. Pour vérifier si un numéro de sécurité sociale est valide, on additionne le nombre n_1 formé par les 13 premiers chiffres au nombre n_2 formé par les 2 derniers chiffres, et on vérifie que c'est un multiple de 97. Par exemple, le numéro 2 55 08 14 168 025 38 est un numéro de sécurité sociale valide car $2550814168025 + 38 = 26297053279 \times 97$.
Écrivez une fonction `ssid_valide(s)` qui prend en argument une chaîne de caractères s et qui renvoie `True` si s est un numéro de sécurité sociale valide. Par exemple, `ssid_valide('255081416802538')` renvoie `True`.

```python
def ssid_valide(s):
    x = int(s[:13]) + int(s[13:])
    return (x % 97 == 0)
```

5. Vous avez intercepté le numéro de sécurité sociale chiffré suivant :

   1010111101011011011111011101111110000010101011010011111111010

   et vous savez que la personne à qui il appartient est un homme né en janvier 98 dans les Alpes-Maritimes — autrement dit, le numéro de sécurité sociale commence par « 1980106 ». Enfin, vous savez que le message est chiffré avec une clé k sur 32 bits.

   Saurez-vous retrouver le numéro de sécurité sociale ainsi que la clé k ?

```
>>> nss = '1010111101011011011111011101111110000010101011010011111111010'
>>> début_nss = nss[:32] # les 32 premiers bits de nss
>>> début = '1980106'
>>> def décimale(s):
...     c=0
...     résultat=''
...     for i in range(len(s)):
...         c = 2*c +int(s[i])
...         if i%4 == 3:
...             résultat=résultat + str(c)
...             c=0
...     return résultat
```

Supposons que l'on connaisse les 8 premiers chiffres du numéro de sécurité sociale. Cela nous donnerait 32 bits (car chaque chiffre est codé sur quatre bits). Notons `début_binaire` ces 32 premiers bits, on aurait alors le résultat suivant : `début_binaire`$\oplus k$ == `début_nss`. Ce qui nous donnerait immédiatement `début_binaire`$\oplus$`début_nss` == k[4] Ainsi, connaissant `début_binaire` et `début_nss`, on peut rapidement trouver la clé `k`! Malheureusement on ne connait que les 7 premiers chiffres de `début`, ce qui nous laisse 10 possibilités.

```
>>> for i in range(10):
...     début_décimale = début+str(i)
...     début_binaire = binaire(début_décimale)
...     print(i,':',début_décimale,début_binaire)
0 : 19801060 00011001100000000001000001100000
1 : 19801061 00011001100000000001000001100001
2 : 19801062 00011001100000000001000001100010
3 : 19801063 00011001100000000001000001100011
4 : 19801064 00011001100000000001000001100100
5 : 19801065 00011001100000000001000001100101
6 : 19801066 00011001100000000001000001100110
7 : 19801067 00011001100000000001000001100111
8 : 19801068 00011001100000000001000001101000
9 : 19801069 00011001100000000001000001101001
```

Comment trouver la clé valide parmi ces 10 ? On va essayer de décoder `nns` avec chacune d'entre elle et voir si cela nous donne un numéro de sécurité sociale valide.

4. Le lecteur peut rapidement de convaincre que pour trois bits a, b et c on a toujours $(a \oplus b = c) \Leftrightarrow (c \oplus a = b) \Leftrightarrow (b \oplus c = a)$

```python
for i in range(10):
    print("= Essai numéro",i, 65*"=")
    début_décimale = début+str(i)
    début_binaire = binaire(début_décimale)
    clé = chiffrement(début_binaire,début_nss)
    print(début_décimale,début_binaire,"clé =",clé)
    ssid_binaire_complet = chiffrement(nss,clé)
    ssid_décimale_complet = décimale(chiffrement(nss,clé))
    print("ssid potentiel :",ssid_binaire_complet)
    if ssid_valide(ssid_décimale_complet):
        print("\n"+ssid_décimale_complet+" est un numéro de sécurité sociale valide\n")
    else:
        print(ssid_décimale_complet,"n'est pas un numéro de sécurité sociale valide\n")
```

```
= Essai numéro 0 ================================================================
19801060 00011001100000000001000001100000 clé = 10110110110110110110110110111111
ssid potentiel : 00011001100000000001000001100000001101000111011001010010000 1
198010603476521 n'est pas un numéro de sécurité sociale valide

= Essai numéro 1 ================================================================
19801061 00011001100000000001000001100001 clé = 10110110110110110110110110111110
ssid potentiel : 00011001100000000001000001100001001101000111011001010010000 1
198010613476521 n'est pas un numéro de sécurité sociale valide

= Essai numéro 2 ================================================================
19801062 00011001100000000001000001100010 clé = 10110110110110110110110110111101
ssid potentiel : 00011001100000000001000001100010001101000111011001010010000 1

198010623476521 est un numéro de sécurité sociale valide

= Essai numéro 3 ================================================================
19801063 00011001100000000001000001100011 clé = 10110110110110110110110110111100
ssid potentiel : 00011001100000000001000001100011001101000111011001010010000 1
198010633476521 n'est pas un numéro de sécurité sociale valide

= Essai numéro 4 ================================================================
19801064 00011001100000000001000001100100 clé = 10110110110110110110110110111011
ssid potentiel : 00011001100000000001000001100100001101000111011001010010000 1
198010643476521 n'est pas un numéro de sécurité sociale valide

= Essai numéro 5 ================================================================
19801065 00011001100000000001000001100101 clé = 10110110110110110110110110111010
ssid potentiel : 00011001100000000001000001100101001101000111011001010010000 1
198010653476521 n'est pas un numéro de sécurité sociale valide

= Essai numéro 6 ================================================================
19801066 00011001100000000001000001100110 clé = 10110110110110110110110110111001
ssid potentiel : 00011001100000000001000001100110001101000111011001010010000 1
198010663476521 n'est pas un numéro de sécurité sociale valide

= Essai numéro 7 ================================================================
19801067 00011001100000000001000001100111 clé = 10110110110110110110110110111000
ssid potentiel : 00011001100000000001000001100111001101000111011001010010000 1
198010673476521 n'est pas un numéro de sécurité sociale valide

= Essai numéro 8 ================================================================
19801068 00011001100000000001000001101000 clé = 10110110110110110110110110110111
ssid potentiel : 00011001100000000001000001101000001101000111011001010010000 1
198010683476521 n'est pas un numéro de sécurité sociale valide

= Essai numéro 9 ================================================================
19801069 00011001100000000001000001101001 clé = 10110110110110110110110110110110
ssid potentiel : 00011001100000000001000001101001001101000111011001010010000 1
198010693476521 n'est pas un numéro de sécurité sociale valide

```

Exercice 6 — Tapis : le retour ! (⋆⋆)

À faire chez vous : refaire les tapis du TP 2 mais en utilisant uniquement des boucles `for` (donc sans `while`) et sans utiliser de fonctions auxilaires (donc vous êtes obligés de faire une boucle `for` dans une boucle `for`).

```python
def tapis_a(largeur,hauteur):
    for i in range(hauteur):
        for j in range(largeur):
            étoile()
        nouvelle_ligne()

def tapis_b(largeur,hauteur):
    for i in range(hauteur):
        for j in range(largeur):
            if j%2==0: # colonne pair
                étoile()
            else: # colonne impair
                dièse()
        nouvelle_ligne()


def tapis_c(largeur,hauteur):
    for i in range(hauteur):
        for j in range(largeur):
            if i%2==0: # ligne pair
                if j%2==0: # colonne pair
                    étoile()
                else:      # colonne impair
                    dièse()
            else: # ligne impair
                if j%2==0: # colonne pair
                    dièse()
                else:      # colonne impair
                    étoile()
        nouvelle_ligne()


def tapis_d(largeur,hauteur):
    for i in range(hauteur):
        for j in range(largeur):
            if i%3==0: # première ligne
                if j%2==0: # colonne pair
                    étoile()
                else:      # colonne impair
                    dièse()
            elif i%3==1: # seconde ligne
                if j%2==0: # colonne pair
                    dièse()
                else:      # colonne impair
                    étoile()
            else: # troisième ligne
                étoile()
        nouvelle_ligne()
```