Cours 08 : Programmation orientée-objet en Python – v0

Connexion à Capytale :

- **login.capytale.fr** (choisir Connexion sans ENT, en bas colonne de droite)

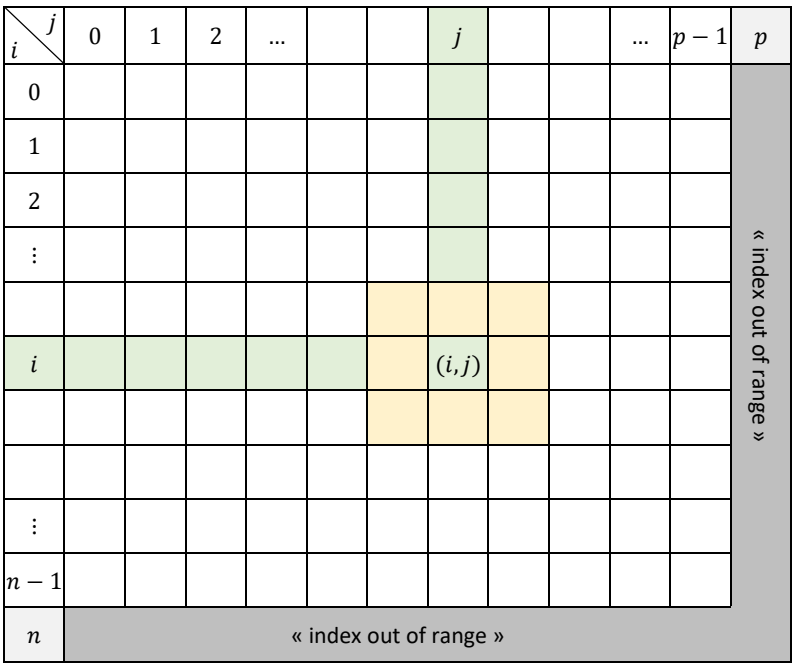
Notebook pour ce cours :

- Outils Python : caaf-3184957
- Notebook-prof (pour accéder aux démos faites au tableau) : c1c6-3316654

Jeu de la vie

- Présentation : https://fr.wikipedia.org/wiki/Jeu_de_la_vie
- Démo : https://www.palais-decouverte.fr/fr/explorer-nos-contenus/experiences-dinformatique/le-jeu-de-la-vie

On modélise la grille du jeu de la vie par un tableau d'entiers à deux dimensions, les cellules vivantes étant représentées par l'entier 1, les autres cellules contiennent la valeur zéro.



Chaque cellule, si elle n'est pas située sur un bord de la grille est entourée de huit cellules voisines.

Ces voisines ont été coloriées en jaune ci-contre, pour une cellule située en (i, j) dans le cas où
$0 < i < n - 1$
et
$0 < j < p - 1$.

Chaque cellule de la grille est repérée par un couple de « coordonnées », (i, j), où i et j sont, respectivement les indices de la ligne et de la colonne à l'intersection desquelles la cellule se trouve.
Si la grille est représentée en machine par un tableau à deux dimensions `T`, la valeur de la cellule repérée par le couple (i, j) est donnée par `T[i][j]`, valant 1 si la cellule est vivante, et zéro sinon.

I) Définition de l'interface

On modélise la grille du jeu de la vie par un tableau d'entiers à deux dimensions, les cellules vivantes étant représentées par l'entier 1, les autres cellules contiennent la valeur zéro.

1) Constructeur de la classe `JeuV`
Le constructeur de la classe aura pour paramètres, `nb_lignes` et `nb_colonnes`.
Chaque objet de la classe sera caractérisé par trois attributs :

- `n` : un entier représentant le nombre de lignes de la grille ;
- `p` : un entier représentant le nombre de colonnes de la grille ;
- `grille` : un tableau à deux dimensions de taille $n \times p$, initialement rempli de zéros.

2) Méthode `affiche`.
On définit une méthode `affiche`, affichant à l'aide de la fonction `print`, les lignes de la grille, les unes au-dessous des autres.

Par exemple, si le jeu est représenté par l'objet `jeu_test` et que la grille est de taille 3×5, sans cellule vivante, on aura :

```
>>> jeu_test.affiche()
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
```

3) Méthode `mise_a_jour0`.

On rappelle que la fonction `random` du module `random` est une fonction ne prenant pas d'argument qui renvoie un flottant tiré au hasard dans l'intervalle $[0; 1[$:

```
>>> from random import random
>>> random()
0.3749425561370866
```

Cette fonction permet de définir une fonction `alea10(p)` renvoyant l'entier 1 avec une probabilité p (où $0 < p < 1$) et 0 avec la probabilité $1 - p$:

```
def alea10(p):
    if random() <= p:
        return 1
    else:
        return 0
```

On définit alors, pour les objets de la classe `JeuV`, la méthode suivante, `mise_a_jour0`, qui affecte au hasard, grâce à la fonction `alea10`, à chacune des cellules de coordonnées $(0,0)$, $(0, 1)$, $(1,0)$ et $(1, 1)$, la valeur 1 avec une probabilité $p = 0{,}5$.

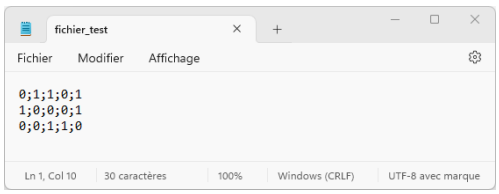
```
class JeuV:
    ...
    def mise_a_jour0(self):
        for i in range(2):
            for j in range(2):
                self.grille[i][j] = alea10(0.5)
```

4) Méthode `charge_grille0`.

Le format **.csv** est un format permettant d'enregistrer le contenu d'un tableau à deux dimensions sous la forme d'un fichier texte.

Par exemple le tableau d'entiers ci-dessous à gauche sera codé par le fichier ci-dessous à droite :

	A	B	C	D	E
1	0	1	1	0	1
2	1	0	0	0	1
3	0	0	1	1	0



Le fichier visible à droite, nommé **fichier_test.csv**, s'il est situé dans le répertoire de travail du programme qui cherche à l'ouvrir, peut-être ouvert à l'aide de Python, par exemple, grâce à la séquence d'instructions suivante :

```
with open("fichier_test.csv", encoding='utf-8') as fd:
    liste_lignes = fd.readlines()
```

On peut créer un fichier **.csv** à l'aide d'un tableur quelconque, pourvu que l'on choisisse un format **.csv** au moment de faire « Enregistrer sous ». L'encodage peut varier selon le système d'exploitation de la machine utilisée, il faudra tester les encodages suivants `"utf-8"`, `"utf-8-sig"` ou `"cp-1252"` si vous créez vous-même le fichier. Le séparateur de colonnes, « ; » sur l'exemple, peut aussi varier et être, par exemple, une virgule « , » ou une tabulation « \t ».

On récupère alors le contenu du fichier sous la forme d'une unique chaîne de caractères, nommée ici `contenu`.

Pour le fichier en exemple, on récupère la liste de chaînes de caractères suivante dont chaque élément correspond à une ligne du fichier **.csv** :

```
['0;1;1;0;1\n', '1;0;0;0;1\n', '0;0;1;1;0\n']
```

On peut alors reconstruire le tableau avec la séquence d'instructions suivante :

```
tab = [ligne.strip().split(";") for ligne in liste_lignes]
```

où l'appel à la méthode `.strip` retire le caractère de fin de ligne `"\n"` et l'argument de la méthode `.split` est le séparateur de colonne, `";"` sur l'exemple.
On a alors sur l'exemple :

```
>>> tab
[['0', '1', '1', '0', '1'], ['1', '0', '0', '0', '1'], ['0', '0', '1', '1',
'0']]
```

Dans le cas où les données à récupérer dans le tableau sont des valeurs entières, il reste alors à convertir les chaînes de caractères (`'0'` ou `'1'` sur l'exemple) en valeurs de type `int`, ce qui peut se faire de la façon suivante, par application de la fonction de conversion de type `int` et l'utilisation de compréhensions de listes :

```
tab_def = [[int(tab[i][j]) for j in range(len(tab[0]))] for i in range(len(tab))]
```

On obtient alors, sur l'exemple, le tableau suivant :

```
>>> tab_def
[[0, 1, 1, 0, 1], [1, 0, 0, 0, 1], [0, 0, 1, 1, 0]]
```

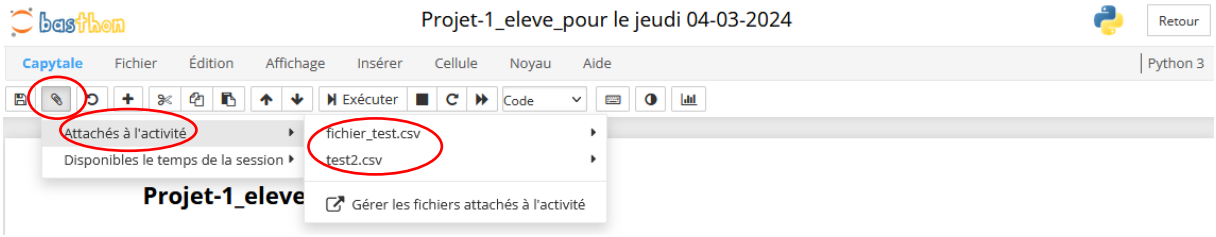
Afin de charger une grille, on pourra alors définir la méthode suivante (que l'on modifiera ultérieurement) :

```
class JeuV:
    ...
    def charge_grille0(self):
        with open("fichier_test.csv", encoding='utf-8') as fd:
            liste_lignes = fd.readlines()
        tab = [ligne.strip().split(";") for ligne in liste_lignes]
        tab_def = [[int(tab[i][j]) for j in range(len(tab[0]))]
                                                for i in range(len(tab))]
        self.grille = tab_def
```

Annexe : Chargement et téléchargement de fichiers annexés à un notebook

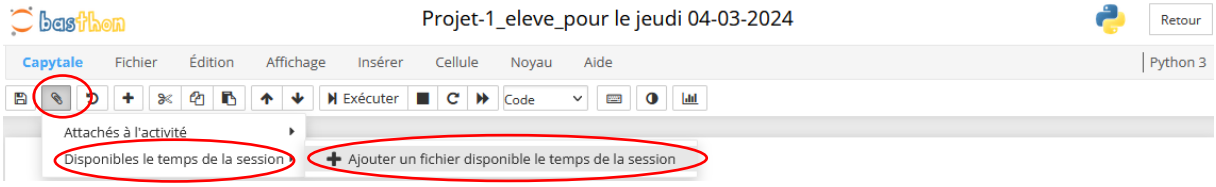
- des fichiers sont annexés au notebook pour les exercices.

On peut les voir grâce au menu suivant (icône « trombone ») :



Les fichiers-texte (**.txt** ou **.csv**) sont accessibles directement depuis le notebook en lecture.

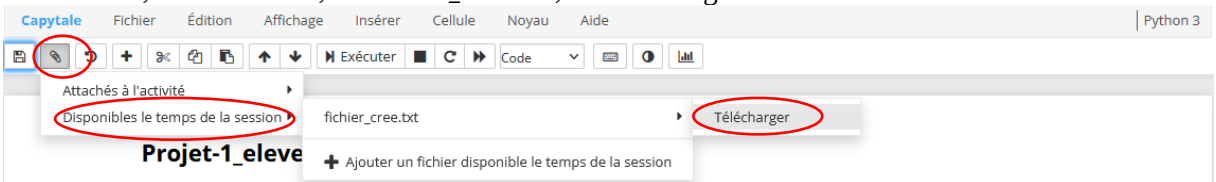
- il est possible de charger un fichier directement pendant que l'on travaille dans un notebook.



Il faut alors cliquer sur « + » pour rechercher sur fichier sur la machine sur laquelle s'exécute le notebook et le charger. Le fichier sera alors disponible le temps de la session, mais pas au-delà, même en enregistrant le notebook. Il faudra le recharger lorsque l'on ouvrira à nouveau le notebook ou si on l'actualise.

- il est possible de télécharger un fichier écrit à l'aide de Python.

Dans ce cas, le fichier créé, ici **fichier_cree.txt**, est téléchargeable *via* le menu suivant :



Travail demandé : à faire dans le notebook 9d6e-3319160 pour le jeudi 04/03/2024.

- lien vers le notebook : https://capytale2.ac-paris.fr/web/c/9d6e-3319160

Q1. Définir la classe `JeuV` et son constructeur, conformément aux spécifications du **I)1)** (fait en cours)

Q2. Définir une instance de la classe, nommée `jeu_vie0`, dont la grille sera de taille 4×5, et vérifier que ses trois attributs ont la valeur attendue, en les affichant sans mise en forme particulière.

Q3. Compléter la définition de la classe `JeuV` en lui ajoutant la méthode `affiche`, agissant comme décrit au **I)2)**.

Q4. Tester la méthode `affiche` avec une instance quelconque, `jeu_vie1`, de la classe modifiée.

Q5. Compléter la définition de la classe `JeuV` en lui ajoutant la méthode `mise_a_jour0`, telle que donnée au **I)3)**.

Q6. Tester la méthode `mise_a_jour0` en l'appelant cinq fois sur une instance quelconque, `jeu_vie2`, créée avec la classe modifiée, et en affichant la grille après chaque appel.

Q7. Compléter la définition de la classe `JeuV` en lui ajoutant la méthode `charge_grille0` présentée au **I)4)**. **On modifiera la définition de la méthode pour vérifier à l'aide d'assertions** (avec `assert`) que le tableau lu dans le fichier a bien les mêmes dimensions que la grille de l'objet auquel on l'applique (représenté par `self` dans la définition de la méthode).

Q8. Afin de pouvoir tester la méthode `charge_grille0`, créer une nouvelle une instance, `jeu_vie3`, de la classe `JeuV`, bien choisie, et tester alors la méthode `charge_grille0` en vérifiant son bon fonctionnement par l'affichage la grille de `jeu_vie3`.

Q9. Définir une méthode `charge_grille`, sur le modèle de la méthode `charge_grille`, mais prenant en paramètre un nom de fichier **.csv**, `nomfichier`, et un encodage des caractères, `encoding`, sous la forme de chaînes de caractères, et :

- vérifiant que le fichier lu correspond bien à un tableau à deux dimensions ;
- redéfinissant les attributs `n` et `p`, et `grille`, de façon à correspondre aux caractéristiques du tableau lu dans le fichier.

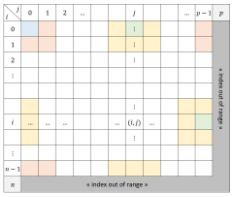
Q10. Tester la méthode avec le fichier **test2.csv**, annexé au notebook, puis avec un fichier **montest.csv**, que vous aurez créé vous-même à l'aide d'un tableur. On laissera apparent l'affichage de la grille résultante et on enregistrera le fichier.

Questions complémentaires (non exigées et non évaluées pour le jeudi 04/03/2024)

Q11. Définir une méthode `liste_voisines_hors_bord`, de paramètres i et j et renvoyant, sous la forme d'une liste de couples, la liste des coordonnées des cellules voisines d'un cellule de coordonnées (i, j), uniquement pour les cellules non situées sur le bord de la grille. On vérifiera, à l'aide d'`assert`, que les valeurs de `i` et `j` correspondent bien à une cellule non située sur un bord.

Q12. Définir une méthode `liste_voisines1`, de paramètres i et j et renvoyant, sous la forme d'une liste de couples, la liste des coordonnées des cellules voisines d'un cellule de coordonnées (i, j), en omettant les cellules voisines qui seraient situées hors du plateau pour les cellules situées sur un bord.

Q13. Définir une méthode `liste_voisines2`, de paramètres i et j et renvoyant, sous la forme d'une liste de couples, la liste des coordonnées des cellules voisines d'un cellule de coordonnées (i, j), en prenant comme voisines pour les cellules d'un bord les cellules situées sur le bord opposé si nécessaire.



Q14. Définir une méthode `mise_a_jour`, faisant appel à une l'une des méthodes renvoyant des listes de voisines pour toutes les cellules de la grille, et actualisant la grille conformément aux règles du jeu de la vie, selon lesquelles une cellule morte devient vivante si exactement trois cellules voisines sont vivantes, tandis qu'une cellule vivante le reste si deux ou trois cellules, parmi ses voisines, sont vivantes et meurt sinon.