TP Python

Version 1

Clément LACAÏLE

janv. 05, 2020

Contents:

Clément LACAÏLE - Groupe 3 SI2

CHAPITRE 1

TP1 : Prise en main

Ce TP a pour but de se familiariser avec le langage Python. Puisque c'est un langage avec lequel j'ai déjà de bonnes bases (je l'ai étudié lors d'un semestre en mobilité internationale), j'ai décidé d'aller un peu plus loin et d'ajouter les deux particularités : l'utilisation de la convention *PEP8* et l'outil de compilation de documentation *Sphinx*.

TP1.**exercice1**()
: Le but de cet exercice est d'afficher « Hello, world ! » dans la console

TP1.**exercice2**()
: Fonction principale de l'exercice 2 : éditeur de fichiers

  Le but de cet exercice est de proposer un outil de gestion de fichier au tra vers d'un menu console tel que :

  1. Charger le fichier
  2. Ajouter du texte au fichier
  3. Lire le fichier chargé
  4. Vider le fichier
  5. Quitter l'outil

  Pour ce faire, j'ai d'abord créé un drapeau (run_flag) à Vrai qui permet, lorsqu'il passe à Faux (par l'option 5), de quitter le programme. Ensuite, pour l'option 1, le chargement du fichier ne charge en fait que le nom du fichier, mais n'uti lise pas de open(). Cette fonction est seulement utilisée pour l'option 2, 3, ou 4, afin de spécifier le mode de lecture/écriture du fichier. En 2 (ajout de texte à la fin du fichier), l'option vaut « a », et les lignes sont ajoutées avec la fonction write(). En 3 (lecture du fichier), le fichier est ouvert avec l'option « r », et lu avec la fonction readlines(). Enfin, en 4, l'open() est utilisé avec l'option « w ». A la fin de chacune de ces options, le fichier est fermé avec la fonction close().

TP1.**exercice3A**()
: Le but de cet exercice est de créer une classe date et de surcharger les opé rateurs = et <.

TP1.**exercice3B**()
: Le but de cet exercice est de créer une classe Etudiant et de pouvoir charger des objets à partir d'un fichier CSV. Comme dans l'exercice 1, on ouvre le fichier CSV avec la fonction open(). Ensuite, on utilise un csv.reader() pour lire les lignes et les colonnes (séparées par des ;). Pour chaque ligne du fichier, on ajoute à la liste des étudiants un nouvel étudiant dont l'age a été parsée a partir d'une date de naissance puis calculée. Enfin, on imprime à l'écran le dernier étudiant inséré, puis le nombre de ligne traitées.

TP1.**input_menu**()
Cette fonction imprime « Entrez un choix » dans la console et retourne la chai ne saisie par l'utilisateur. Elle vérifie que l'entrée est conforme au dicti onnaire défini gloabelement en début de fichier.

Renvoie the string input by the user

Type renvoyé string

TP1.**load_file**(*file_name*)
Cette fonction charge un fichier en mode écriture et place le curseur à la f in. Si le fichier n'existe pas, il est créé.

Paramètres **file_name** (*str*) – the name of the file to open or create

Renvoie the file opened

Type renvoyé file

TP1.**print_dict**(*dict*)
Cette fonction imprime un dictionnaire dans la console

Param : dict (Dictionary) : the dictionary to print

Exemple

```
>>> print_dict({"1": "Hello,", "2": "world!"})
1 : Hello,
2 : world!
```

class ex3_class_Date.**Etudiant**(*nom*, *age*)
Classe de définition d'un étudiant

Tests :

```
>>> import ex3_class_Date
>>> etu1 = ex3_class_Date.Etudiant("Pierre", 18)
>>> etu1.getEmail()
Pierre@etu.univ-tours.fr
```

class ex3_class_Date.**Ma_Date**(*year*, *month*, *day*)
Cette classe définie une classe date qui hérite de la classe date intégrée à Python. Les méthodes __eq__ et __lt__ ont été surchargées.

ex3_class_Date.**calcul_age**(*date_naiss*)
Calcul de l'age d'un individu à partir de sa date de naissance.

Paramètres **date_naisse** (Ma_Date) –

Renvoie (int) l'age de l'individu

Lève **TypeError si le paramètre n'est pas de type Ma_Date** –

ex3_class_Date.**parse_date**(*date_to_parse*)
Parse une date au format jj/mm/aaaa en un objet Ma_Date Params :

date_to_parse (str) la date à parser au format jj/mm/aaaa

Renvoie Ma_Date la date parsée

CHAPITRE 2

TP2 : TKinter

Le but de ce TP est de se familiariser avec la création d'interface graphique en Python en utilisant la librairie TKinter.

TP2.**clear**()
: Fonction de gestion du bouton « Clear »

  Cette fonction permet de vider complètement le buffer. Elle met à jour l'afficheur en conséquence.

TP2.**delpress**()
: Fonction de gestion du bouton « Del »

  Cette fonction permet de gérer le comportement du bouton « Del ». Il permet d'effacer le dernier caractère saisi par l'utilisateur. Elle met à jour l'afficheur en conséquence.

TP2.**equalpress**()
: Fonction de gestion du bouton « = »

  Cette touche permet de gérer le comportement du bouton « = ». Elle évalue le contenu de la saisie avec la fonction eval() puis affiche le résultat dans l'afficheur. Si une exception est levée (par exemple en cas de division par zéro), l'afficheur prend la valeur ERROR et le buffer est vidé.

TP2.**press**(*num*)
: Fonction de gestion de la pression d'un bouton de valeur « num »

  Cette fonction permet de gérer l'appui d'un touche (sauf « = », Clear et Del). Elle met à jour le contenu de l'afficheur. :param num : :type num : str

CHAPITRE 3

TP3 : Exceptions et chiffrement

Le but de ce TP est de se familiariser avec la gestion des exceptions en Python ainsi que le chiffrement. Pour cela, j'ai adapté le code de l'éditeur de fichier vu dans le TP1 afin qu'il gère la connexion d'utilisateurs par mot de passes chiffrés à l'aide de la librairie bcrypt. Ensuite, j'ai modifié toutes les fonctions et autres lignes afin qu'elles puissent, dans le cas où celà était nécessaire, lever des exceptions, ou les gérer

TP3.**create_user**()
: Fonction de création d'un utilisateur

  Demande à un utilisateur d'entrer un nom et un mot de passe du nouvel utilisateur à créer puis l'ajoute au fichier « users » en le hashant. Si le fichier n'existe pas, il est créé.

TP3.**exercice1**()
: Fonction principale de l'exercice 1 : éditeur de fichiers

  Le but de cet exercice est de proposer un outil de gestion de fichier sécurisé qui reprend les fonctionnalités développées au TP1, en y ajoutant une authentification et la gestion d'exceptions.

TP3.**hash_password**(*password_to_hash*)
: Hashage du password

TP3.**input_menu**()
: Cette fonction imprime « Entrez un choix » dans la console et retourne la chai ne saisie par l'utilisateur. Elle vérifie que l'entrée est conforme au dicti onnaire défini gloabelement en début de fichier.

  Renvoie the string input by the user

  Type renvoyé string

TP3.**load_file**(*file_name*)
: Cette fonction charge un fichier en mode écriture et place le curseur à la f in. Si le fichier n'existe pas, il est créé.

  Paramètres **file_name** (*str*) – the name of the file to open or create

  Renvoie the file opened

  Type renvoyé file

TP3.**login**()
: Fonction de connexion

  Demande à l'utilisateur un nom et un mot de passe et vérifie que le couple est bien présent dans le fichier « users ».

Renvoie True si le couple est bien présent dans le fichier « users », False sinon

TP3.**print_dict**(*dict*)

Cette fonction imprime un dictionnaire dans la console

Param : dict (Dictionary) : the dictionary to print

Exemple

```
>>> print_dict({"1": "Hello,", "2": "world!"})
1 : Hello,
2 : world!
```

CHAPITRE 4

TP4 : Matplotlib

Le but de ce TP est de prendre en main l'outil Matplotlib qui permet de dessiner des graphiques dans une fenêtre, en reprenant une syntaxe similaire à celle de Matlab/Scilab. Le TP est composé de 6 questions qui sont présentées dans les sous-parties suivantes

TP4.**genere_flt_normal**(*moy*, *stdev*)
Retourne un float entre min et max selon une loi normale. :param moy : la moyenne de la répartition :type moy : float :param stdev : l'écart-type de la répartition :type stdev : float

Renvoie (float) une valeur aléatoire selon une loi normale de paramètres (moy, stdev)

TP4.**genere_flt_uniform**(*min*, *max*)
Retourne un float entre min et max selon une loi uniforme. :param min : la borne inférieure :type min : float :param max : la borne supérieure :type max : float

Renvoie (float) une valeur aléatoire entre min et max

TP4.**genere_int_uniforme**(*min*, *max*)
Retourne un entier entre min et max selon une loi uniforme :param min : la borne inférieure :type min : int :param max : la borne supérieure :type max : int

Renvoie (int) une valeur aléatoire entre min et max

TP4.**plot_linear**(*var*, *equation*, *title=''*)
Draws a given equation using pyplot

Paramètres
— **a numpy linspace instance**(*var*) –
— **the equation using var**(*equation*) –
— **the title of the graph**(*title*) –

TP4.**question1**()
Le but de cette question est de générer un ensemble de nombres aléatoires. Pour cela, on fixe un nombre maximum de valeurs à générer max=500 ainsi qu'une seed. Ensuite, on génère des nombres suivants une loi uniforme grâce à la fonction genere_flt_uniform() définie dans ce module, puis des nombres suivants une loi normale grâce à la fonction genere_flt_normal() définie dans ce module. Ces fon ctions utilisent la librairie random de Python. L'utilisation de la fonction numpy.linspace remplace en une seule ligne la génération de ces données.

TP4.**question2**(*unif_x*, *unif_y*, *norm_x*, *norm_y*, *should_plot=True*)

Le but de cette question est de générer un graphique à partir des données générées dans la question1. Pour cela, on utilise les fonctions plot et show de la librairie matplotlib. Pour la fonction plot, le premier argument est la liste des valeurs d'abscisse tandisque le second est la liste des valeurs en ordonnée. La fonction show() ne prend pas de paramètre.

TP4.**question34**(*serie_unif_X*, *serie_unif_Y*, *serie_norm_X*, *serie_norm_Y*, *should_plot=True*)

Le but de cette question est d'afficher plusieurs courbes avec styles et couleurs variés ainsi que modifier les noms des axes, la légende, ajouter des flèches pour montrer des zones.

Pour ce faire, j'ai décidé de réutiliser les 4 séries de données créés dans les questions précédentes et de diviser l'affichage de la fenêtre avec 4 « sous-graphiques », grâce à la fonction subplots(nb_lignes, nb_colonnes, partage_des_valeurs). Cette fonction retourne un objet figure ainsi que 4 objets représentants les graphiques individuels. Le paramètre de partage des valeurs en ordonnée (sharey) permet aux graphiques de fixer les mêmes bornes pour les axes y (au lieu des bornes automatiques), afin de pouvoir comparer plus facilement les répartitions. A gauche, les graphiques portent sur les données réparties uniformément, tandis que ceux de droite portent sur les données réparties normalement. Pour ces répartitions, on retrouve un nuage de point créé avec la fonction scatter(X, Y, couleur_marqueur, forme_marqueur) ainsi qu'un histogramme (hist(Y, couleur)) montrant clairement le type de répartition. J'ai ajouté des étiquettes fléchées avec la fonction annotate(titre, position, vecteur_fleche, style_fleche), un titre par colonnes (set_title()), et enfin un titre principal (suptitle()).

TP4.**question5**(*should_plot=True*)

Le but de cette question est d'afficher un camembert (l'histogramme a déjà été traité dans la question précédente).

On crée une liste de 4 parts aléatoires selon une loi uniforme entre 0 et 25% ainsi qu'une liste de noms des parts qui seront affichés sur le camembert. Ensuite, on utilise la fonction pie(parts, noms)

TP4.**question6**(*should_plot=True*)

Le but de cette question est d'afficher une surface 2D dans un espace 3D (mesh). Pour cela, on génère des valeurs entre -5 et 5 pour deux vecteurs : X et Y en utilisant la fonction linspace issue de numpy. Ensuite on créé deux matrices de coordonnées à partir des vecteurs X et Y en utilisant la fonction numpy.meshgrid. Enfin on créé la matrice des valeurs que l'on veut tracer : ici on a choisi de tracer la fonction $sqrt{X^2+Y^2}$. Enfin, on trace la forme dans un graphique 3D en utilisant les valeurs génères précé- demment.

TP4.**random**() → x in the interval [0, 1).

CHAPITRE 5

TP5 : SQLite

Le but de ce TP est de mettre en place une base de données avec SQLite. L'exécutable *sqlite3* est déjà présent dans le dossier de ce TP, il est inutile de le réinstaller. En outre, j'ai décidé de ne pas utiliser la librairie ORM SQLAlchemy, car c'est une librairie que je connais déjà. Je perds du temps, mais cela m'a permis de manipuler plus en profondeur le langage.

TP5.**parse_csv**(*csv_name*)

EXERCICE 1 : lecture des fichiers csv

Cette fonction permet de lire les fichiers communes, départements et régions.csv Les fichiers CSV comportent des entêtes sur 8 lignes. Le parser s'en débarasse donc.

Paramètres **le nom du fichier csv à parser** (*csv_name*) –

Renvoie un tableau à deux dimensions contenant pour chaque ligne une liste de champs

TP5.**peuple_table_commune**(*db_conn*, *dao*, *info_communes*)

EXERCICE 1 : Fonction de peuplement de la table Commune de la base de données

Cette fonction permet, à partir d'un tableau à deux dimensions, de peupler la table Commune de la base de données. Le paramètre info_communes est une liste de listes contenant les informations des communes recueillies grâce à la fonction parse_csv().

Paramètres **info_communes** (*[[string]]*) –

TP5.**peuple_table_departement**(*db_conn*, *dao*, *info_depts*)

EXERCICE 1 : Fonction de peuplement de la table Departement de la base de données

Cette fonction permet, à partir d'un tableau à deux dimensions, de peupler la table Departement de la base de données. Le paramètre info_depts est une liste de listes contenant les informations des départements recueillies grâce à la fonction parse_csv().

Paramètres **info_depts** (*[[string]]*) –

TP5.**peuple_table_region**(*db_conn*, *dao*, *info_regions*)

EXERCICE 1 : Fonction de peuplement de la table Region de la base de données

Cette fonction permet, à partir d'un tableau à deux dimensions, de peupler la table Region de la base de données. Le paramètre info_regions est une liste de listes contenant les informations des régions recueillies grâce à la fonction parse_csv().

Paramètres **info_regions** (*[[string]]*) –

class Commune.**Commune**(*code_commune*, *nom_commune*, *pop_totale*, *code_dept*)

Cette classe représente une commune

Attr : _code_commune (int) : l'identifiant commune (PK) _nom_commune (str) : le nom de la commune _pop_totale (int) : le nombre d'habitants _code_dept (int) : pseudo clé étrangère sur les département

class Commune.**CommuneDAO**

Cette classe permet de créer la table « commune » dans la base de données, mais aussi d'insérer un élément, supprimer un élément, lire un élément, lire un élément selon son département, lire tous les éléments de la table et enfin mettre à jour un élément.

create(*db_conn*)

Fonction de création de la table « commune » dans la base spécifiée par l'argument db_conn

Paramètres **db_conn** (*Connection*) –

Renvoie True si la création s'est effectuée correctement, False sinon

delete(*db_conn*, *code_commune*)

Fonction de suppression d'un élément dans la base de données

Paramètres
- **db_conn** (*Connection*) –
- **code_commune** (*int*) –

Renvoie True si la suppression s'est correctement déroulée, False sinon

insert(*db_conn*, *commune*)

Fonction d'insertion d'un objet Commune dans la base de données

Paramètres
- **db_conn** (*Connection*) –
- **commune** (Commune) –

Renvoie True si l'insertion s'est correctement déroulée, False sinon

read(*db_conn*, *code_commune*)

Fonction de lecture d'un élément dans la base de données

Cette fonction permet de lire un élément « Commune » dans la base de données en fonction de son code_commune, identifiant primaire de la table. Il est important de noter que cette fonction fait appel à la méthode cursor.fetchall()qui renvoie une liste d'éléments répondant à la clause WHERE spécifiée dans la requête. L'utilisation de cursor.fetchone() n'a pas été jugée prudente. Ici, puisque code_commune est l'identifiant unique de la table, la liste retrounée contiendra toujours au plus 1 élément.

Paramètres
- **db_conn** (*Connection*) –
- **code_commune** (*int*) –

Renvoie (list) liste d'éléments vérifiants le code_commune, None sinon

read_all(*db_conn*)

Fonction de lecture de tous les éléments dans la base de données

Cette fonction permet de lire tous les éléments « Commune » dans la base de données.

Paramètres **db_conn** (*Connection*) –

Renvoie (list) liste d'éléments, None si aucun élément n'est présent dans la base.

read_by_dept(*db_conn*, *code_dept*)

Fonction de lecture d'un élément dans la base de données

Cette fonction permet de lire un élément « Commune » dans la base de données en fonction de son code_dept.

Paramètres
- **db_conn** (*Connection*) –
- **code_dept** (*int*) –

Renvoie (list) liste d'éléments vérifiants le code_dept, None sinon

update(*db_conn*, *code_commune*, *nom_commune*, *pop_totale*, *code_dept*)
Fonction de mise à jour d'un élément dans la base de données

Paramètres
- **db_conn** (*Connection*) –
- **code_commune** (*int*) –
- **nom_commune** (*str*) –
- **pop_totale** (*int*) –
- **code_dept** (*int*) –

class Departement.**Departement**(*code_dept*, *nom_dept*, *code_region*)
Cette classe représente un département

Attr : _code_dept (int) : code du département, sert de PK _nom_dept (string) : nom du département _code_region (int) : code de la région à laquelle appartient le department

class Region.**Region**(*code_region*, *nom_region*)
Cette classe représente un région

Attr : _code_region (int) : code de la région, sert de PK _nom_region (string) : nom de la région

CHAPITRE 6

TP6 : Numpy/Scipy

Le but de ce TP est de prendre en main les librairies Numpy et Scipy.

TP6.**question1**()
: Le but de la question est de créer un tableau de dimension 3 avec un shape de (4, 3, 2) remplit avec des nombres aléatoires.

  Pour y répondre, on génère d'abord un tableau de 24 nombres aléatoires à l'aide de la fonction numpy.randint(). Ensuite, on les ré-arrange dans un tableau à trois dimensions avec la fonction numpy.reshape(4,3,2). Enfin, on affiche ses attributs ndim, shape, size, dtype, itemsize, data.

TP6.**question2**()
: Le but de la question est de créer 2 matrices 3x3 initialisées avec les entiers de 0 à 8 pour la 1e et de 2 à 10 pour la 2e puis calculer le produit des 2 (différence entre * et dot). Transposer une matrice.

  On génère les valeurs des deux matrices m1 et m2 à l'aide de la fonction linspace() puis on forme les dimensions avec reshape(). On calcule ensuite le produit terme à terme des deux matrices avec l'opérateur ".". On transpose ensuite la matrice issue de m1.dot(m2)

TP6.**question3**()
: Cette question a pour but de calculer le déterminant et l'inverse d'une matrice, résoudre un système d'équations linéaires, puis calculer les valeurs et vecteurs propres d'une matrice.

  D'abord, on instancie une matrice m1 telle que : m1 = [[10, 9, 1], [9, 10, 5], [1, 5, 9]] Ensuite, on calcule son déterminant à l'aide de la fonction np.linalg.det puis son inverse à l'aide de la fonction np.linalg.inv. Ensuite, on résoud le système m1 = [-50, 40, 180] à l'aide de np.linalg.solve. Enfin, on détermine les valeurs et vecteurs propres de m1 avec la fonction np.linalg.eig(m1)

TP6.**question4**(*should_plot=True*)
: Le but de cette question est d'pprocher un ensemble de points par une courbe (optimize.curve_fit ou interpolate.interp1d).

  On génère d'abord un ensemble de points suivants une loi exponentielle avec une « marge » aléatoire uniforme de + ou - 0.5, puis on affiche ces points dans un nuage avec matplotlib. Ensuite, on fit la courbe avec la fonction scipy.optimisze.curve_fit. Cette fonction prend comme paramètres la fonction dont les paramètres sont à fitter (ici, une fonction exponentielle tq f(T) = A*exp(1/T)) ainsi que les données à partir desquelles fitter la courbe. Enfin, on affiche la courbe sur un graphe.

TP6.**question5**()
Le but de cette question est de lire une image jpeg et afficher l'image originale et réduite en taille. Cette question n'est pas traitée.

CHAPITRE 7

TP7 : Numpy/Scipy

Le but de ce TP était de créer un serveur web et de pouvoir afficher du contenu en fonction de la connexion ou non d'un utilisateur présent dans un fichier. J'ai décidé de mettre en place une application Flask très simple, remplissant le même rôle.

CHAPITRE 8

Indices and tables

— genindex
— modindex
— search

Index des modules Python

Index

T

U