Aide-mémoire PYTHON3 - v3.5.2

Juillet 2016

1. **Fonctions**
   - int(x) : convertit x, en entier
   - float(x) : convertit x, int ou str, en réel
   - str(x) : convertit x, int ou float, en str
   - list(x) : convertit x en liste
   - tuple(x) : convertit x en tuple
   - dict(x) : convertit la séquence de couples x en dictionnaire
   - set(x) : convertit x en ensemble
   - help(x) : aide sur x
   - dir(x) : liste des attributs de x
   - type(x) : type de x
   - print(...) : imprime
   - input(x) : imprime le string x et lit le string qui est introduit au clavier
   - round(x [,ndigits]) : valeur arrondie du float x à ndigits chiffres (par défaut 0)
   - len(s) : longueur de la séquence s
   - range([start], stop, [step]) : retourne une suite arithmétique d'entiers

2. **Gather, scatter et keyword arguments**
   - def fun(*args) : *gather* de la liste des arguments transmis dans args
   - fun(*s) : *scatter de la séquence s lors de l'appel
   - def fun(**keyw) et fun(a=3, b=5) : keyw est le dictionnaire `{'a':3, 'b':5}`
   - ayant `d = {'a':3, 'b':5}`, f(**d) mappe les éléments de d avec les paramètres formels (a=3 et et b=5)

3. **Modules**
   - math : accès aux constantes et fonctions mathématiques (pi, sin(), sqrt(x), exp(x),floor(x) (valeur plancher), ceil(x) (valeur plafond), ...) : exemple : math.ceil(x)
   - copy : copy(s), deepcopy(s) : *shallow* et *deepcopy* de s
   - sys : argv : liste des arguments passés lors de l'exécution du script (argv[0] donne le nom du script lui même)
   - pickle ou json :
     - dumps(v) : sérialise v
     - loads(r) : reconstitue l'objet
     - dump(v,f) : sérialise v, mis dans le fichier f
     - load(f) : reconstitue à partir de la sérialisation lue sur f
   - shelve
     - db = shelve.open() : créer un fichier, objet shelve
     - db.close() : fermeture

4. **Opérations et méthodes sur les séquences (str, list, tuples) :**
   - min(s),max(s) : élément minimum, maximum
   - sum(s) : (ne fonctionne pas pour les string) : somme de tous les éléments (valeur numérique)
   - s.index(value, [start, [stop]]) : premier indice de value dans s[start :stop]
   - s.count(sub [,start [,end]]) : nombre d'occurrences sans chevauchement de sub dans s[start :end]

5. **Méthodes sur les str :**
   - s.lower() : string avec caractères en minuscule
   - s.upper() : string avec caractères en majuscule
   - s.islower(), s.isdigit(), s.isalnum(), s.isalpha(), s.isupper() : vrai si dans s on a (respectivement) des minuscules, des chiffres, des car. alphanumériques, alphabétiques, majuscules
   - s.find(sub [,start [,end]]) : premier indice de s où le sous string sub est trouvé dans s[start :end]
   - s.replace(old, new[, co]) : copie de s en remplaçant toutes les (ou les co premières) occurrences de old par new.
   - s.format(...) : copie de s après formatage (sert en particulier pour les outputs)
   - s.capitalize() : copie de s avec la première lettre en majuscule
   - s.strip() : copie de s en retirant les *blancs* en début et fin
   - s.join(t) : créer un str qui est le résultat de la concaténation des éléments de la séquence de str t chacun séparé par le str s
   - s.split([sep [,maxsplit]) : renvoie une liste d'éléments séparés dans s par le caractère sep (par défaut *blanc*) ; au maximum maxsplit séparations sont faites (par défaut l'infini)

6. **Opérateurs et méthodes sur les listes :**
   - s.append(v) : ajoute un élément valant v à la fin de la liste
   - s.extend(s2) : rajoute à s tous les éléments de la liste s2
   - s.insert(i,v) : insert l'objet v à l'indice i
   - s.pop([i]) : supprime l'élément d'indice i de la liste (par défaut le dernier) et retourne la valeur de l'élément supprimé
   - s.remove(v) : supprime la première valeur v dans s
   - s.reverse() : retourne la liste, le premier et dernier élément échangent leurs places, le second et l'avant dernier, et ainsi de suite
   - s.sort(key=None, reverse=False) : trie s en place
   - del s[i], del s[i :j] : supprime un ou des éléments de s
   - zip (a,b), zip(a,b,c), ... : construit une liste de couples, resp. triples, ..., dont le ième élément reprend le ième élément de chaque séquence a, b [,c]
   - it = iter(s) : créé un itérateur qui pourra être utilisé avec next(it) qui donne élément suivant de l'itérateur s'il existe, exception StopIteration sinon
7. **Méthodes sur les dict :**
   - d.clear() : supprime tous les éléments de d
   - d.copy() : *shallow* copie de d
   - .fromkeys(s,v) : créée un dict avec les clés de s et la valeur v
   - d.get(k [,v]) : renvoie la valeur d[k] si elle existe v sinon
   - d.items() : liste des items (k,v) de d
   - d.keys() : liste des clés
   - d.pop(k [,v]) : enlève d[k] s'il existe et renvoie sa valeur ou v sinon
   - d.popitem() : supprimer un item (k,v) et retourne l'item sous forme de tuple
   - d.setdefault(k [,v]) : d[k] si elle existe sinon v et rajoute d[k]=v
   - d.update(s) : s est une liste de tuples que l'on rajoute à d
   - d.values() : liste des valeurs de d
8. **Méthodes sur les set :**
   - s = set(v) : initialise s comme étant un set contenant les valeurs de v
   - s.add(v) : ajoute l'élément v au set s (ne fait rien s'il y est déjà)
   - s.clear() : supprime tous les éléments du set s
   - s.remove(v) : supprime l'élément v du set (erreur si v n'est pas présent dans s)
   - s.discard(v) : supprime l'élément v du set (pas d'éventuel message d'erreur)
   - s.pop() : supprime et renvoie un élément arbitraire de s
9. **Méthodes sur les fichiers :**
   - f=open('fichier') : ouvre 'fichier' en lecture
   - f=open('fichier','w') : ouvre 'fichier' en écriture
   - f=open('fichier','a') : ouvre 'fichier' en écriture en rajoutant après les données déjà présentes
   - f.read() : retourne le contenu du fichier f
   - f.readline() : lit une ligne
   - f.readlines() : renvoie la liste des lignes de f
   - f.write(s) : écrit la chaîne de caractères s dans le fichier f
   - f.close() : ferme f
10. **Exceptions :**
   - ```
     try:
         ...
     raise ...
         ...
     except:
         ...
     else:
         ...
     finally:
         ...
     ```
11. **Classes :**
   - ```
     class MyClass(objects):
         ...
     def __init__(self, ...):
         ...
     def __str__(self):
         ...
     def __repr__(self):
         ...
     def my_fun(self, ...):
         ...
     ```