Cahier de notes

Notes personelles

Rémy Guyonneau - remy.guyonneau@univ-angers.fr

Dernière mise à jour: 24 mai 2020

Table des matières

1 Notes sur le langage Python

1.1 Commande utiles

1.1.1 le terminal python

Lancer l'interpreteur :

```
python3
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Quitter l'interpréteur

```
>>> exit()
```

1.1.2 La commande dir()

La fonction dir() retourne toutes les propriétés et méthodes d'un objet donné en paramètre. Sans paramètre en entrée elle permet notamment d'accéder à tous les packages disponibles.

```
1 dir() # Lister tous les imports disponibles a ce moment du code
```

Exemple d'utilisation :

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
'__package__', '__spec__']', '__package__', '__spec__']
>>> import tkinter
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
'__package__', '__spec__', 'tkinter']
```

```
1 dir(name) # Lister les proprietes/methodes disponibles pour "name"
```

Exemple d'utilisation :

```
>>> dir(tkinter)
['ACTIVE', 'ALL', ...,  'Text', 'Tk', 'TkVersion', ...]
>>> dir(tkinter.Tk)
['_Misc__winfo_getint', '_Misc__winfo_parseitem', '__class__', ..., 'after',
'after_cancel', 'after_idle', 'anchor', 'aspect', 'attributes', 'bbox', ...]
```

1.1.3 La commande help()

Elle permet d'accéder à l'aide python. Elle prend un argument, qui correspond à l'objet pour lequel on veut obtenir des informations.

```python
help(name) # Obtenir de l aide sur le module "name"
```

Exemple d'utilisation :

```
>>> help(tkinter.Tk)
>>> help(list)
```

Pour quitter l'aide on peut utiliser la touche 'q' du clavier.

1.1.4 Gestion de grands nombres

Il est possible d'utiliser l'underscore pour améliorer la visibilité de grands nombres :

```python
un_grand_nombre = 10_000_000_000
un_plus_petit_grand_nombre = 2_000_000_000
somme = un_grand_nombre + un_plus_petit_grand_nombre
print(f"{somme:_}")
```

Ce qui donne l'affichage suivant

```
12_000_000_000
```

1.1.5 La commande type()

Il est possible d'utiliser la commande type() pour connaître le type d'une variable. Par exemple

```python
variables = [1, 3.2, lambda x: x + 1, (1, 2), [1, 2]]
for var in variables:
    print(type(var))
```

Donne l'affichage suivant

```
<class 'int'>
<class 'float'>
<class 'function'>
<class 'tuple'>
<class 'list'>
```

1.1.6 Les variables non utilisées

La convention en python pour la gestion des variables non utilisées est de les appeler "_". Des exemple :

```python
mon_tuple = (1, 2, 3)
val1, _, val2 = mon_tuple
print(f"val1:{val1}, val2:{val2}")
```

Affiche

```
val1:1, val2:3
```

La deuxième valeur du tuple n'est pas utilisée.

1.1.7 todo ?

pip list

which python

1.2 Environnements virtuels

1.2.1 Pourquoi ?

Un espace pour installer des paquets spécifiques à un ou plusieurs projets

1.2.2 Créer un environnement avec venv (standard)

```
python3 -m venv test_env
```

Remarque : le "-m" c'est pour dire qu'on lance le module venv. Le module venv attend un argument : le nom de l'environnement (test_env) ici.

1.2.3 Activer l'environnement

```
source test_env/bin/activate
```

1.3 Listes

1.3.1 Accéder à des éléments

Pour accéder aux derniers éléments d'une liste :

```
1 ma_liste = [1,2,3,4]
2 print(ma_liste[-1]) # affiche 4
3 print(ma_liste[-2]) # affiche 3
```

Attention, cette notation aussi peut générer une exception "IndexError".

1.3.2 Parcourir des listes

Pour parcourir une liste sans utiliser d'indice

```
1 ma_liste = [1, 2,"test"]
2 for val in ma_liste:
3     print(val)
```

Affiche :

```
1
2
test
```

Pour parcourir une liste tout en comptant les éléments :

```python
ma_liste = [1, 2, "test"]
for idx, val in enumerate(ma_liste):
    print(f"ma_liste[{idx}] = {val}")
```

Affiche :

```
ma_liste[0] = 1
ma_liste[1] = 2
ma_liste[2] = test
```

Il est aussi possible de commencer à compter à partir d'une certaine valeure :

```python
ma_liste = [1, 2, "test"]
for idx, val in enumerate(ma_liste, start=1):
    print(f"{idx} : {val}")
```

Affiche :

```
1 : 1
2 : 2
3 : test
```

1.3.3 Parcourir deux listes en même temps

Il est possible d'utiliser la fonction zip() pour parcourir plusieurs listes. Un exemple avec deux listes, mais peut être étendu à plus de listes :

```python
list1 = [1, 2, 3]
list2 = ["un", "deux", "trois"]

for val1, val2 in zip(list1, list2):
    print(f"{val1} : {val2}")
```

```
1 : un
2 : deux
3 : trois
```

1.4 Tuples

1.4.1 Récupérer les valeurs

Récupérer toutes les valeurs :

```python
mon_tupple = (1, 2, 3)
a, b, c = mon_tupple
print(f"a:{a}, b:{b}, c:{c}")
```

```
a:1, b:2, c:3
```

Récupérer une partie des valeurs d'un tuple. Ce qu'il ne faut pas faire :

```python
mon_tupple = (1, 2, 3, 4, 5)
a, b, c = mon_tupple
print(f"a:{a}, b:{b}, c:{c}")
```

```
    a, b, c = mon_tupple
ValueError: too many values to unpack (expected 3)
```

Ce qu'il est possible de faire :

```python
mon_tupple = (1, 2, 3, 4, 5)
a, b, *c = mon_tupple
print(f"a:{a}, b:{b}, c:{c}")
```

```
a:1, b:2, c:[3, 4, 5]
```

1.5 Saisies

1.5.1 Récupérer une saisie sans l'afficher à l'écran (terminal)

Cas d'utilisation :

```python
nom = input("Utilisateur: ")
psw = input("Mot de passe: ")
print("Connexion...")
```

```
Utilisateur: remy
Mot de passe: mdp
Connexion...
```

On voit que le mot de passe s'affiche sur le terminal, ce qui n'est pas désirable...

À la place, il est possible de faire :

```python
from getpass import getpass
nom = input("Utilisateur: ")
psw = getpass("Mot de passe: ")
print("Connexion...")
```

```
Utilisateur: remy
Mot de passe:
Connexion...
```

Ce qui est quand même mieux...

1.6 Classes

1.6.1 Définir un attribut dynamiquement

Cas simple sans attribution dynamique du nom des attributs :

```python
# declaration d une classe "vide"
class Ma_classe():
    pass
# instanciation de la classe dans l objet inst
inst = Ma_classe()
# on ajoute un attribut "att1" a la classe et lui affecte la valeur "coucou"
inst.att1 = "coucou"
# on ajoute un attribut "att2" a la classe et lui affecte la valeur "coucou"
inst.att2 = 3
# affichage des valeurs des attributs
print(inst.att1, inst.att2)
```

```
coucou 3
```

Le même exemple mais avec une attribution dynamique des noms des attributs

```python
# declaration d une classe "vide"
class Ma_classe():
    pass

# instanciation de la classe dans l objet inst
inst = Ma_classe()
# on passe par des variables pour les noms des attributs
att1_name = "att1"
att2_name = "att2"
# on ajoute un attribut "att1" a la classe et lui affecte la valeur "coucou"
setattr(inst, att1_name, "coucou") # objet, nom, valeur
# on ajoute un attribut "att2" a la classe et lui affecte la valeur 3
setattr(inst, att2_name, 3) # objet, nom, valeur
# affichage des valeurs des attributs
print(inst.att1, inst.att2)
```

```
coucou 3
```

Un autre exemple d'utilisation :

```python
# declaration d une classe "vide"
class Ma_classe():
    pass

# instanciation de la classe dans l objet inst
inst = Ma_classe()

# defintion des attributs dans un dictionnaire
mes_attributs = {"att1":"coucou", "att2":3}

# ajout des attributs a la classe
for key, value in mes_attributs.items():
    setattr(inst, key, value)

# affichage des valeurs des attributs
print(inst.att1, inst.att2)
```

```
coucou 3
```

```python
# declaration d une classe "vide"
class Ma_classe():
    pass

# instanciation de la classe dans l objet inst
inst = Ma_classe()

# defintion des attributs dans un dictionnaire
mes_attributs = {"att1":"coucou", "att2":3}

# ajout des attributs a la classe
for key, value in mes_attributs.items():
    setattr(inst, key, value)

# recuperation et affichage des attributs
for key in mes_attributs.keys():
    print(getattr(inst, key))
```

```
coucou
3
```

1.7 Gestion de fichiers/ressources

1.7.1 Context Manager

Dès qu'une ressource demande une ouverture puis une fermeture (e.g. des fichiers), il est possible d'utiliser un context manager pour ne pas avoir à gérer la fermeture.

Par exemple, à la place de faire :

```python
f = open("tmp.txt", "r")
file_contents = f.read()
f.close()
print(file_contents)
```

```
Contenu du fichier tmp.txt
Sur plusieurs lignes...
```

Il est possible de faire (ce qui donne le même résultat qu'au dessus)

```python
with open("tmp.txt", "r") as f:
    file_contents = f.read()
print(file_contents)
```

Le context manager peut être utilisé dès qu'il faut ouvrir et fermer des ressources (pour un lock dans un thread par exemple, pour l'accès à des base de données...).

1.8 Fonctions Lambda / Fonctions sans noms