RÉPUBLIQUE
FRANÇAISE

Liberté
Égalité
Fraternité

100537 : PYTHON PERFECTIONNEMENT

PARTIE 2//2

Institut de la Gestion publique et du Développement économique

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Objectifs de la formation

- Après la phase d'initiation (stage Python 8487), ce module a pour but:

- De vous sensibiliser sur les bonnes pratiques Python.
- De vous faire découvrir l'éventail des modules Python.
- De vous permettre de délivrer du code efficace et plus robuste.
- De maitriser les différentes notions structurantes de Python

- En résumé:
- « D'aller un peu plus loin dans votre apprentissage »

Objectifs pédagogiques :

- ❑ Maitriser le langage Python
- ❑ Adopter des bonnes pratiques
- ❑ Avoir large vision des possibilités de Python
- ❑ Etre autonome

Le programme

Jour 1

Réactivation

Plus loin avec Les classes.

Les itérateurs

Organiser son projet (et jour 2)

Jour 2

Documentation du projet

Gestion des fichiers structurés

IHM

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Présentation du formateur, des stagiaires et du stage

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Réactivation

Séquence 1

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Sommaire

- ❑ Rappels

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Plus loin avec les classes

Séquence 2

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

IGPDE

Sommaire

- ❑ Les dataclasses
- ❑ Manipulations des attributs
- ❑ Surcharge d’un type de base

Les dataclasses

dataclasses.ipynb

- Rappel: les tuples nommés (module collections)

Les dataclasses permettent de prendre en charge la génération des constructeurs.

```
class Demo():
    def __init__(self, nom, prenom, age):
        self.nom = nom
        self.prenom = prenom
        self.age = age
    def __str__(self):
        return(f'{self.prenom} - {self.nom.upper()} - {self.age}')
```

```
people = Demo('dupont', 'paul', 18)
print(people)
```

```
paul - DUPONT - 18
```

Avec le décorateur @dataclass

```
from dataclasses import dataclass
```

```
@dataclass
class Demo():
    nom: str
    prenom:str
    age:int
```

```
people = Demo('dupont', 'paul', 18)
print(people)
```

```
Demo(nom='dupont', prenom='paul', age=18)
```

Manipulations des attributs

- Rappel: le cheminement de la recherche d'un attribut
- La méthode __getattr__ est appelée en dernier recours

Manip_attr.ipynb

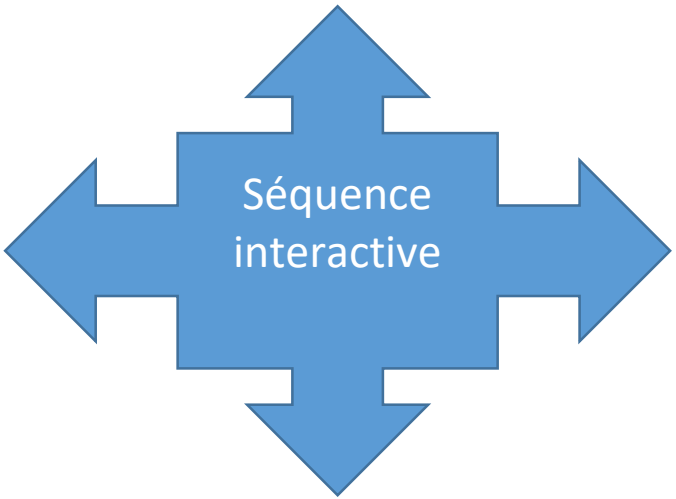
Défi: Modifier une classe pour accéder aux attributs quelque soit la casse

Surcharge d'un type élémentaire

Défi : réaliser une classe liste dont les éléments sont accessibles par un index numerique ou sous forme de chaine:
Exemple liste[1] ou liste['1']

Manip_base.ipynb

```
from collections import UserList

__getitem__(self, key)

__setitem__(self, key, value)
```



RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Itérateurs

Séquence 3

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Sommaire

- ❑ Notions : iterable, iterateur
- ❑ Objet iterable
- ❑ Construire un iterateur
- ❑ L’instruction ‘for’
- ❑ Yield
- ❑ Lecture d’un gros fichier

Notions

Des notions imbriquées:
A retenir:

- Un **itérateur** est le composant qui va réaliser la distribution des éléments.
- Un **itérable** est le composant qui fourni les éléments à distribuer

Pour travailler ensemble, ces deux composants utilisent des conventions

Convention

Une structure peut être iterable (fournir des éléments)
si:
Elle implémente la methode ‘__iter__’
OU
Elle peut être parcourue par un index allant de 0 à n via ‘__getitem__’

Un composant est un iterateur (capable de distribuer des éléments d’une structure itérable)
si:
Il implémente la méthode ‘__next__’ **et** ‘__iter__’

A commenter =>

```
a = [1, 5, 7]
a.__iter__() # retourne un iterateur => les listes sont iterables
```

```
<list_iterator at 0x29288e99a88>
```

```
iter(a)
```

```
<list_iterator at 0x29288e9c088>
```

```
print(hasattr(a,'__next__'))
print(hasattr(a, '__iter__'))
```

```
False
True
```

Un objet iterable

```
]: class  MonIterable():
       def __getitem__(self,index):
           if index > 5:
               raise StopIteration
           return(chr(index + 65))
```

```
]: b = MonIterable()
   iterateur = iter(b)
```

```
]: next(iterateur)
```

```
]: 'A'
```

A commenter =>

```
for item in b:
    print(item)
```

```
A
B
C
D
E
F
```

```
print(hasattr(b,'__next__'))
print(hasattr(b, '__iter__'))
```

```
False
False
```

Construire un iterateur

```
class MonIterateur():
    def __init__(self, iterable):
        self.iterable = iterable
        self.index = 0
    def __next__(self):
        self.index += 1
        if self.index <=  len(self.iterable):
            return self.iterable[self.index -1]
        else:
            raise StopIteration
```

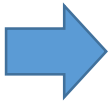
```
a = MonIterateur(['e', 'r', 'i','c'])
```

```
next(a)
```

```
'e'
```

```
for item in a:
    print(item)
```

```
---------------------------------------------------------
TypeError                                 Traceback (mos
<ipython-input-63-7015a0e1ecaf> in <module>
----> 1 for item in a:
      2     print(item)

TypeError: 'MonIterateur' object is not iterable
```



```
def __iter__(self):
        return self
setattr(MonIterateur,'__iter__',__iter__)
```

```
a = MonIterateur(['e', 'r', 'i','c'])
for item in a:
    print(item)
```

```
e
r
i
c
```

L’instruction for : elle émule un iterateur

```
[45]: for chiffre in ob1:
          print(chiffre)

4
3
2
1
```

=

```
44]: it = iter(ob1)
while 1:
    try :
        chiffre  = next(it)
        print(chiffre)
    except StopIteration :
        break

4
3
2
1
```

L’instruction yield

- Une fonction de génération (générateur) est un mode simplifié pour écrire un itérateur
- Il est activé en fonction du besoin
- Utilisation du mot clé ‘yield’ en lieu et place de ‘return’
- Il s’épuise après consommation

```
def generateur():
        yield 1
        yield 2
gen1 = generateur()
for a in gen1:
  print(a)
for a in gen1:
  print(a)
```


```
1
2
```

??

Rechargement du générateur

```
gen2 = generateur()
print(next(gen2))
```

```
1
```

Dynamique

```
def generateur_demo():
    print('DANS le generateur :j entre dans le generateur')
    yield  'donnee 1 envoyée'
    print('DANS le generateur : zone entre 1 et 2')
    yield  'donnee 2 envoyée'
    yield  3
    print('DANS le generateur :je sors du generateur')
```

Sorties

1

```
premier next a venir
DANS le generateur :j entre dans le generateur
donnee 1 envoyée
```

2

```
2eme next a venir
DANS le generateur : zone entre 1 et 2
donnee 2 envoyée
```

3

```
3eme next a venir
3
```

```
next de trop demandé
DANS le generateur :je sors du generateur

---------------------------------------------------------------------------
StopIteration                             T
<ipython-input-7-9ed05cbac4bb> in <module>
      1 print('next de trop demandé')
----> 2 next(ess) ### provoque une erreur

StopIteration:
```

Exécution

```
ess = generateur_demo() ### ici il ne se passe rien de particulier
```

```
print('premier next a venir')
print(next(ess))
```

```
print('2eme next a venir')
print(next(ess))
```

```
print('3eme next a venir')
print(next(ess))
```

```
print('next de trop demandé')
next(ess) ### provoque une erreur
```

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Exemples

```
def generateur_demo2(liste):
    while 1:
        for item in liste:
            yield item
```

```
a = ['janvier', 'fevrier', 'mars']
ess2 = generateur_demo2(a)
for _ in range(10):
    print(next(ess2))
```

```
janvier
fevrier
mars
janvier
fevrier
mars
janvier
fevrier
mars
janvier
```

```
def generateur_demo3():
    from time import  time
    debut  = time()
    while 1:
        yield time() - debut
```

```
essai = generateur_demo3()
print(next(essai))
for _ in range(10000000):
    pass
print(next(essai))
for _ in range(20000000):
    pass
print(next(essai))
for _ in range(100000):
    pass
print(next(essai))
```

```
0.0
0.6029300689697266
1.829085350036621
1.8437409400939941
```

Domaine d'utilisation des générateurs (yield) :

- Traitement de structure (fichier/flux) présentant un risque de saturation mémoire (performance – Deep Learning).
- Besoin de persistance de variables au sein d'une fonction: un 'return' libère les variables locales.
- Traitement sur des portions d'une suite d'éléments (fenêtrage).

Lire un gros fichier

manip_gros_fic.ipynb

```
with open('bigfic.avi', 'rb') as input:
    rbytes =  input.read()
    with open('sortie.avi', 'wb') as output:
        output.write(rbytes)
```

Utilisation d'une variable qui charge globalement de le fichier

```
with open('bigfic.avi', 'rb') as input, open('sortie.avi', 'wb') as output:
    output.write(input.read())
```

Dialogue direct entre le reader et le writer

```
def lecture_chunk():
      file = open('bigfic.avi', 'rb')
      while True:
         data = file.read(400000)

         if data:
              yield data
         else:
             break

with  open('sortie.avi', 'wb') as output:
    for segment in lecture_chunk():
        output.write(segment)
```

Avec une coroutine

Fonctionne dans tous les cas

Organiser son projet

Séquence 4

Sommaire

- ❑ Gestion d’environnements virtuels Python
- ❑ Tests unitaires
- ❑ Le Packaging et la distribution de module Python
- ❑ La documentation

§ voir poetry

Gestion d'environnements virtuels Python

Objectifs:

- Cloisonner et figer une version Python avec ses librairies
- Avoir plusieurs versions de python sur son poste

Python –m venv <repertoire>
(*python -m venv essai_de_version*)

```
Répertoire de C:\Users\egerman01\essai_de_version

13/11/2019  16:29    <DIR>          .
13/11/2019  16:29    <DIR>          ..
13/11/2019  16:29    <DIR>          Include
13/11/2019  16:29    <DIR>          Lib
13/11/2019  16:29               124 pyvenv.cfg
13/11/2019  16:29    <DIR>          Scripts
               1 fichier(s)              124 octets
               5 Rép(s)  918 183 321 600 octets libres
```

Depuis la version 3.6 de Python, le module virtualenv est inclus dans Python.
Avec des versions antérieures faire: pip install virtualenv pour installer le module venv

Utilisation (1/2)

Dans le répertoire Scripts

```
Répertoire de C:\Users\egerman01\essai_de_version\Scripts

13/11/2019  16:29    <DIR>          .
13/11/2019  16:29    <DIR>          ..
13/11/2019  16:29             2 315 activate
13/11/2019  16:29             1 045 activate.bat
13/11/2019  16:29             1 515 Activate.ps1
13/11/2019  16:29               368 deactivate.bat
13/11/2019  16:29            93 065 easy_install-3.7.exe
13/11/2019  16:29            93 065 easy_install.exe
13/11/2019  16:29            93 047 pip.exe
13/11/2019  16:29            93 047 pip3.7.exe
13/11/2019  16:29            93 047 pip3.exe
13/11/2019  16:29           415 248 python.exe
13/11/2019  16:29           414 736 pythonw.exe
```

Activation:
\essai_de_version>Scripts\activate.bat

Avant activation:
where python
C:\Users\xxxx\AppData\Local\Programs\Python\Python37-32\python.exe

Après activation:
where python
C:\Users\xxx\essai_de_version\Scripts\python.exe
C:\Users\xxx\AppData\Local\Programs\Python\Python37-32\python.exe

Utilisation (2/2)

Le prompt est préfixé avec le nom de l'environnement virtuel

(essai_de_version) C:\essai_de_version>python -V
Python 3.7.4

```
\essai_de_version\Lib\site-packages

   .
   ..
26 easy_install.py
   ebcdic
   ebcdic-1.1.1.dist-info
```

Une commande PIP 'install' devient locale à notre environnement

Pour revenir au python natif: *\essai_de_version>Scripts\deactivate.bat*

Pour aller plus loin: pyenv https://github.com/pyenv/pyenv

Tests unitaires

- Ils sont destinés à améliorer la robustesse du code
- Ils s'intègrent dans le processus de fabrication
- Pour tous les langages (junit, jsunit, phpunit etc.)

Pour python:

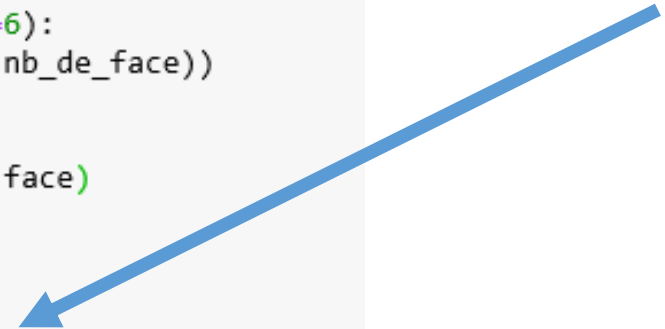
- Unittest (module inclus)
- pytest (à installer)
- Doctest (des tests dans la documentation)

Imports

```python
import unittest
import random

class De:
    def __init__(self, nb_de_face=6):
        self.face = list(range(1, nb_de_face))

    def roule(self):
        face = random.choice(self.face)
        return(face)

if __name__ == "__main__":
    mon_de = De()
    for _ in range(10):
        print(mon_de.roule())
```

```
4
5
4
1
```

Pour un avant-test autonome



Première étape

```
import random
import unittest


class De:
    def __init__(self, nb_de_face=6):
        self.face = list(range(1, nb_de_face))

    def roule(self):
        face = random.choice(self.face)
        return(face)

class MyTest(unittest.TestCase):
    def test_lance(self):
         mon_de = De()
         self.assertTrue(  0 < mon_de.roule() <7)

if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Ajout

Le nom est Préfixé par test_

Ajout

Sous Jupyter la syntaxe d'appel de unittest doit être adaptée.

La syntaxe normale d'appel d'unittest*: unittest.main()*

Mise en pratique

Défi :Compléter les tests pour la classe ‘Dé ’ , vérifier les fréquences des lancés
Aller plus loin : comment faire un dé pipé ?.

Mesure de la couverture des tests

Plus les tests activeront des lignes métiers plus les tests permettront de s'assurer de la robustesse du module.

- Utiliser le module 'coverage' *(pip install coverage)*

Puis:

coverage run test_unit.py

Le sommaire du rapport:

coverage report -m

```
Name          Stmts   Miss  Cover   Missing
-----------------------------------------
lib\de.py         7      0   100%
test_unit.py      8      0   100%
-----------------------------------------
TOTAL            15      0   100%
```

Coverage HTML

Coverage report: 86%

Module ↓	statements	missing	excluded	coverage
lib\depipe.py	20	8	0	60%
test_unit.py	36	0	0	100%
Total	**56**	**8**	**0**	**86%**

coverage.py v4.5.4, created at 2019-11-30 21:28

Utiliser le coverage-defi.py

Coverage for **lib\depipe.py** : 60%

20 statements | 12 run | 8 missing | 0 excluded

```
1  import random
2  from collections import Counter as Counter
3  class De:
4      def __init__(self, nb_de_face=6):
5          self.face = list(range(1, nb_de_face+1))
6
7      def triche_sur(self, face):
8          self.face.append(face)
9
10     def roule(self):
11         face = random.choice(self.face)
12         return(face)
13
14     def affiche(self):
15         proba= round(1/ len(self.face)* 100,2)
16         c =  Counter(self.face)
17         for face in c:
18             print(face, proba * c[face]   )
19
20 if __name__ == '__main__' :
21     mon_de = De()
22     mon_de.affiche()
23     mon_de.triche_sur(2)
24     mon_de.affiche()
25
26
27
28
```

Préparation du packaging Python

§ voir poetry

- Le packaging Python permet d'organiser l'ensemble des modules d'un projet (fichiers .py) en les regroupant au sein d'un ou plusieurs dossiers.
- Un dossier sera défini comme un **package** dès lors qu'il contiendra un fichier **__init__.py**.
- Le fichier**__init__.py** pourra être vide car seule sa présence est importante. Ce fichier permet de restreindre les modules exportés

__init__,py n'est plus obligatoire depuis python >= 3.3
Mais reste une bonne pratique

Les imports

Rappel

- Un package est un dossier contenant un fichier __init__.py (vide ou non vide). Un package regroupe des modules
- L'utilisation spécifique d'une fonction contenue dans un module d'un package se fait par : from package.module import fonction
- L'appel de toutes les fonctions contenues dans un module d'un package se fait par : from package.module import *

Packager pour distribuer

§ voir poetry

Un module est distribué sous 2 formes :
-Une archive source à construire (obtenir un => build)
-Un exécutable (wheel)

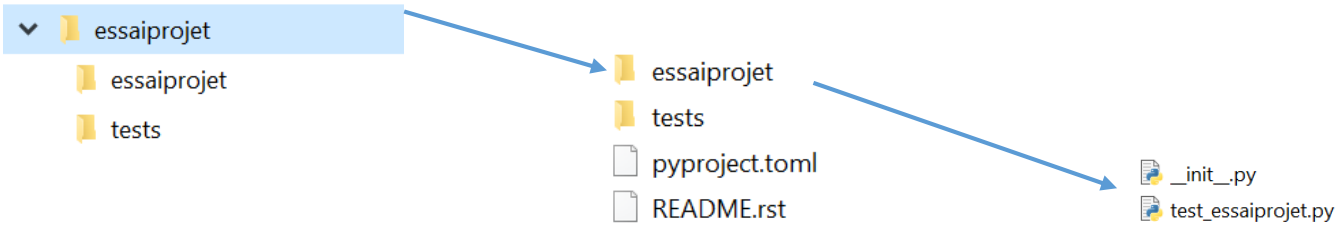
Génération de deux fichiers:
Une archive et un build

Poetry

Poetry permet de commencer ou d'organiser proprement un projet Python par:

- La création d'un répertoire contenant le source du projet, les librairies et le répertoire des tests.
- La fourniture d'outils pour distribuer le module

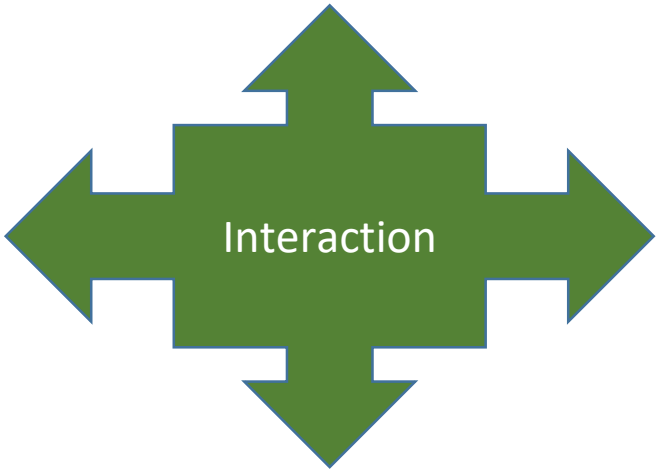
```
poetry new essaiprojet
```



Pourquoi pas vous ?

1) Créer un projet par poetry

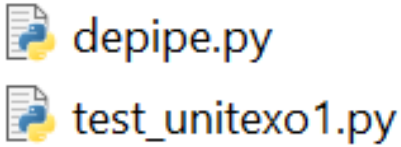
2) Quelles sont les options de poetry ?



3) Garnir les fichiers

4) Packager et distribuer son module

Résultats de la commande



Où vont aller ces fichiers ?
(prendre les fichiers dans le repertoire poetry_igpde)

```
(base) PS C:\Users\33681\Documents\8785> tree    MonModule  /f
Structure du dossier pour le volume OS
Le numéro de série du volume est 00000082 1805:44CB
C:\USERS\33681\DOCUMENTS\8785\MONMODULE
    pyproject.toml
    README.rst

├──monmodule
        __init__.py

└──tests
        test_monmodule.py
        __init__.py
```

Pourquoi pas vous ?

A faire en groupe:

- Examiner le fichier pyproject
- Que mettre dans le README ?
- Installer *(poetry install)*
- Lancer des tests *(poetry run python –munittest discover)*
- Builder *(poetry build) : quel est le résultat ?*

Guide

(1) Création d'un projet:

poetry new MonPackage

(2) Lister les options: lancer la commande poetry sans option

(3) Garnir les fichiers :

Résoudre les dépendances dans les imports

Modifier le fichier: pyproject.toml en enlevant les dépendances de pytest

installer le module: *poetry install*

Lancer les tests par: *poetry run python –munittest discover*

(4) Faire le build : *poetry build*

Manipulation : installer par dans le terminal powershell le package par:

poetry run pip install .\MonModule-0.1.0-py3-none-any.whl

Lancer une console python : poetry run python

Faire l'import du module

Squelette à compléter (notebook: exo_poetry)

Entrée [ ]:
```
# A COMPLETER
from XXXX import XXXX
# FIN
```

Entrée [ ]:
```
mon_de = De()
print('un premier lancé')
print(mon_de.roule())
print( 'un generateur')
def generateur_lance(face):
    _de = De(face)
    while(1):
        yield _de.roule()

# A COMPLETER : instancier un générateur de lancé d'un dé à 6 faces et un autre pour un dé de 8 faces
print('un de avec 6 faces')
de6 =  XXXXXXXXXXXXXXX( X)
print('un autre avec 8 faces')
de8 =  XXXXXXXXXXXXXXX( X)
# FIN

tupple_de = zip(de6, de8)
cp = 0
for item in tupple_de:
    print(item)
    # A COMPLETER : ajouter une condition qui stoppe le traitement quand le premier dé donne un 6 et le deuxieme un 8
    if XXXXXX == XXXXX:
        XXXXXX
    #FIN
    cp += 1
    if cp > 1000:
        print("je crois que vous avez fait une boucle infinie...")
```

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

documenter son projet

Séquence 5

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Sommaire

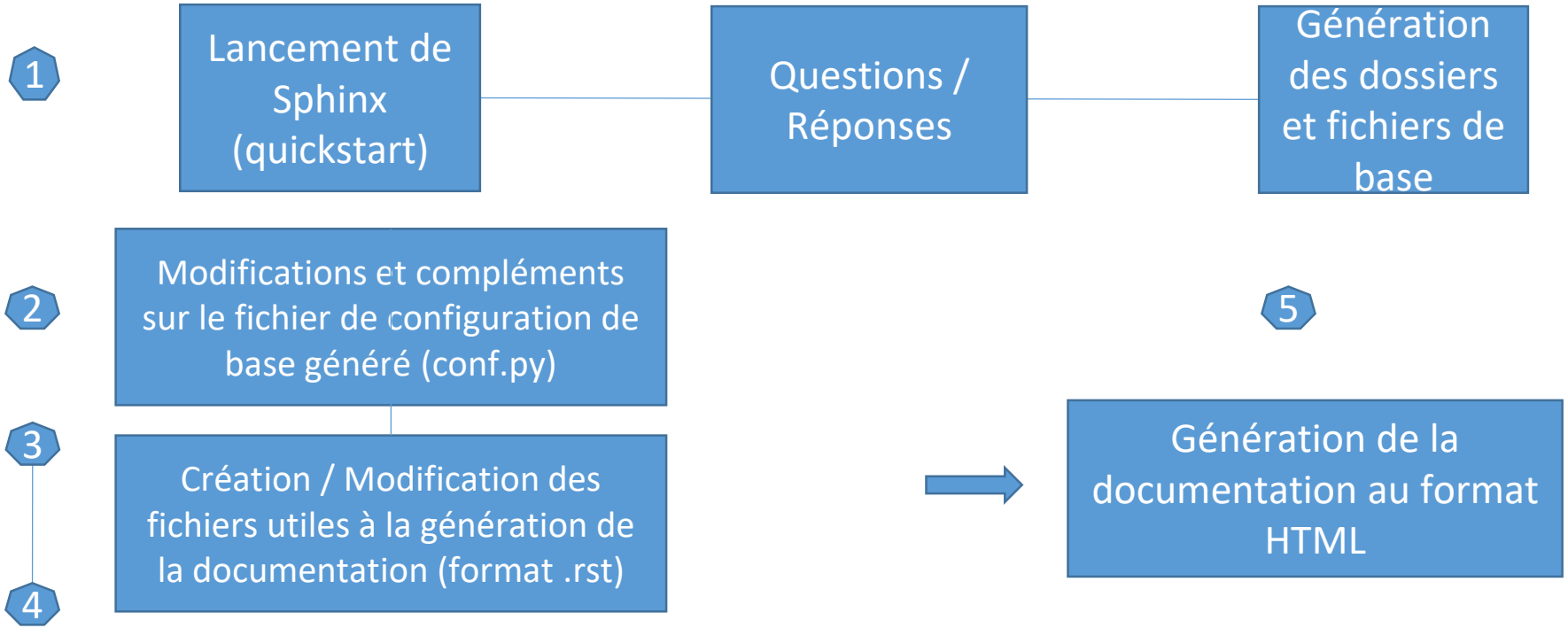
- ❑ Présentation de SPHINX
- ❑ Mise en œuvre

Documenter son projet: SPHINX

- Sphinx-doc est un logiciel qui a su s'imposer très rapidement dans le milieu professionnel comme l'outil indispensable pour générer de la documentation de qualité.
- Outil libre de type générateur de documentation, puissant et disposant d'une large communauté d'utilisateurs
- S’appuie sur des fichiers au format reST (reStructuredText) qu’il convertit au format HTML, PDF, man et autres formats
- Commande d’installation : >> **pip install sphinx**

Processus

- Processus macro de génération de la documentation (format HTML)



1ère Etape : pour commencer

- >> sphinx-quickstart

  - commande qui propose un questionnaire permettant de générer la structure de base des dossiers (build et source) et des fichiers nécessaires à la création automatique de la documentation d'un projet.

La configuration

Questions / Réponses en rouge (**Enter** pour conserver la valeur par défaut, **Y** pour Oui, **N** pour Non, ou **valeur textuelle**)

>**Root path for the documentation**
Indiquez ou créez la documentation. Par défaut, l'emplacement où vous vous trouvez.
>**Separate source and build directories (y/N)** : **Y**
Répondez toujours oui (y) à cette question, toujours dans le but de bien com
>**Name prefix for templates and static dir : Enter**
Permet de stipuler le préfixe pour vos dossiers et fichiers rattachés à Sphinx
>**Project name : Nom du projet**
Saisissez ici le nom de votre projet, en respectant la casse.
>**Author name(s) : Nom de l'auteur**
Saisissez ici les noms des auteurs.
>**Project version** : **1.0**
Indiquez ici la version de votre projet.
>**Project release** : **1.0.0**
Indiquez ici la release de votre version. Par défaut à la même valeur que la v
>**Source file suffix**
Sert à préciser le suffixe des fichiers sphinx-doc. Il est fortement conseiller d
>**Name of your master document (without suffix)**
Permet de préciser comment s'appellera la première page de votre docume « » : **Enter**
>**Do you want to use the epub builder (y/N)** : **Enter**
Si vous désirez pouvoir générer des epubs, placez ce paramètre à oui.
>**autodoc: automatically insert docstrings from modules (y/N)** : Y
Ce paramètre doit être placé à oui, afin de pouvoir documenter du code Python.
>**doctest: automatically test code snippets in doctest blocks (y/N)** : **Enter**
Pas d'info particulières sur ce que fait vraiment ce paramètre.
….

…../

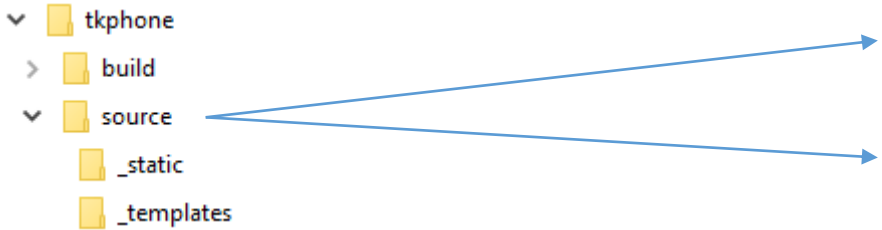
Séparation des sources ?
Auteur ?
Version ?
Etc..

Résultat

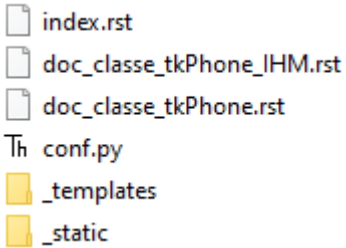
A l'issue de la dernière réponse, la structure de base de la documentation est désormais prête. Vous disposez alors d'un sous-dossier nommé **source** contenant 2 fichiers : **index.rst** et **conf.py** ainsi que deux dossiers nommés « **_static** » et « **_template** »

Un sous-dossier nommé « **build »** sera créé lors de la première demande de génération de la documentation par **l'exécution** du fichier **make.bat** créé à la racine de votre projet.

Exemple



Fichiers et sous-dossiers de « source »



2ème Etape : Modification de la configuration

- Modification du fichier **conf.py** du dossier « **source** »

➢ Dé-commenter les 3 lignes de la section - **Path setup** -

- import os
- import sys
- sys.path.insert(0, os.path.abspath('..')) # modifier le path ‘.’ par ‘..’

➢ Vérifier :

- La présence à minima de la liste **extensions** = ['sphinx.ext.autodoc', 'sphinx.ext.doctest‘] dans la section -General configuration –

Si l’item 'sphinx.ext.autodoc’ n’est pas présent soit l’ajouter directement à la liste soit ajouter la ligne ci-dessous.

- extensions.append('sphinx.ext.autodoc') dans la section - Options for HTML output –

3eme étape : compléter les fichiers RST

- Création du ou des fichiers .rst qui seront déclarés dans le fichier index.rst du dossier « source »

Le format reST impose une syntaxe et une indentation stricte pour chaque élément

Informations minimales de la syntaxe reST nécessaires pour la construction des fichiers .rst :

.. Texte de commentaire (deux point suivi d'un espace + commentaire)

Texte représentant un titre (texte débutant en début de ligne souligné avec le signe =)
==========

.. automodule:: nom du module (deux point suivis d'un espace + nom de la directive suivie de :: et d'un espace + le nom du module)

.. autoclass:: nom de classe (deux point suivis d'un espace + nom de la directive suivie de :: et d'un espace + le nom de la classe)
:members: (3 espaces suivis de : + nom de l'option suivie de : et d'un espace + la valeur de l'option si besoin)

.. literalinclude:: chemin et nom du fichier à inclure (directive d'inclusion d'un fichier à prendre en compte dans la génération de la documentation)

Les directives (macro)

Astuce: lancer la commande : sphinx-apidoc –o source

- Création du ou des fichiers .rst qui seront déclarés dans le fichier index.rst du dossier « source »

Exemple : fichier doc_classe_tkphone.rst

```
Documentation du module Tkphone
===============================

.. automodule:: tkPhone

Le module tkphone est le module principal de l'application "RepTelpeR".
Il permet de gérer un répertoire téléphonique personnel en utilisant
les ressources graphiques proposées par le module TKINTER.

.. autoclass:: Allo
    :members:
    :private-members:
    :special-members:

.. literalinclude:: ../tkPhone.py
```

.. automodule:: est une directive

.. autoclass:: est une directive qui permet d'indiquer le nom de la classe qui sera prise en compte pour générer la documentation

.. literalinclude:: est une directive qui indique le nom du programme .py dans lequel se trouve la classe spécifiée

Exemple

Exemple : fichier doc_classe_tkphone_IHM.rst

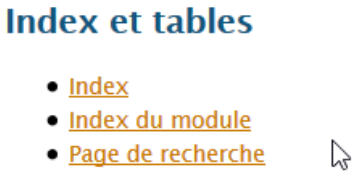
```
Documentation du module Tkphone_IHM
===================================

.. automodule:: tkPhone_IHM

Le module tkphone_IHM permet de créer l'interface graphique de l'application "RepTelpeR".
Les méthodes de ce module sont définies mais ne sont pas opérationnelles.
La partie fonctionnelle est réalisée dans la classe ALLO du module tkPhone.

.. autoclass:: Allo_IHM
    :members:
    :private-members:
    :special-members:

.. literalinclude:: ../tkPhone_IHM.py
```

4eme étape: modification de index.rst

Afficher une table des matières:

Utilisation de la directive **toctree** et de ses **options**

.. toctree:: (deux point suivis d'un espace + nom de la directive suivie de ::)
  :maxdepth: 2 (3 espaces suivis de : + nom de l'option suivie de : et d'un espace + la valeur de l'option)
  (1 ligne vide)
  nom_du_fichier (3 espaces suivi du nom du fichier (rest) sans le suffixe .rst)
  nom_du_fichier (idem pour tous les fichiers rest à intégrer dans la documentation)



Index et tables
==========

* :ref:`genindex` (en début de ligne, une * suivie d'un espace + :ref: + entre guillements le nom du fichier html généré pour la page des index
* :ref:`modindex` (en début de ligne, une * suivie d'un espace + :ref: + entre guillements le nom du fichier html généré pour la page des modules
* :ref:`search` (en début de ligne, une * suivie d'un espace + :ref: + entre guillements le nom du fichier html généré pour la page de recherche

Ces 3 dernières lignes génèrent avec * une liste à puces de liens hypertextes **(:ref:**)
vers les noms-cible spécifiés (page html générées ou créées sans l'extension)

Résultat

```
:orphan:

.. RepTel documentation master file, created by
   sphinx-quickstart on Sat May  4 22:54:21 2019.
   You can adapt this file completely to your liking, but it should at least
   contain the root `toctree` directive.

Bienvenue dans la documentation de *RepTelpeR*
==============================================

*RepTelpeR* est une application de gestion de répertoire téléphonique personnel.
Les données conservées sont le nom, le prénom et le n° de téléphone d'une personne.

.. toctree::
   :maxdepth: 2
   :numbered:
   :caption: Table des matières

   doc_classe_tkPhone
   doc_classe_tkPhone_IHM

.. codeauthor:: Martin MERCET <martin.mercet@dgfip.finances.gouv.fr>

Index et tables
===============

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

.. Lignes de commentaires

Texte à afficher sur la page d'accueil « index.html »

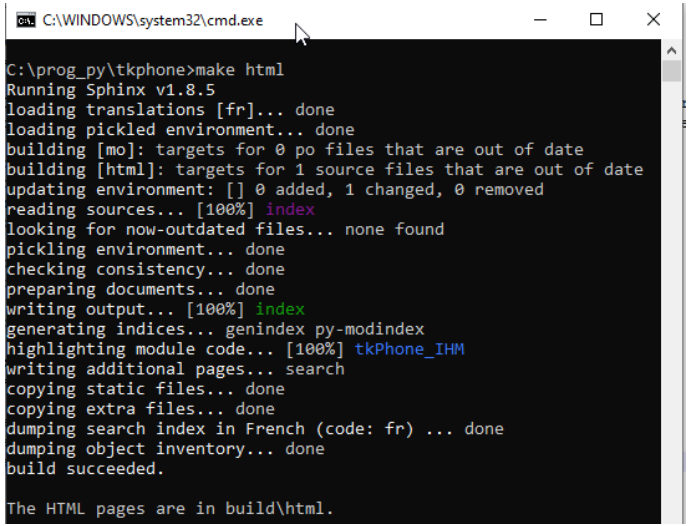
.. toctree:: est une directive de génération de l'arborescence des pages déclarées (fichiers .rst)

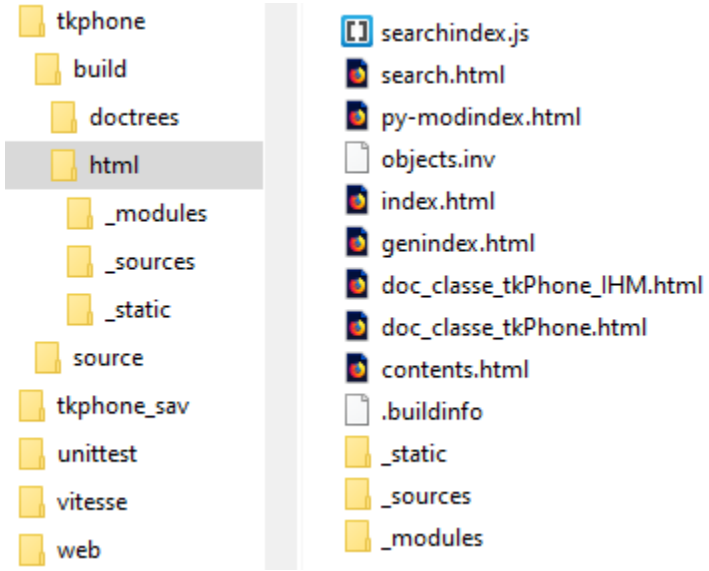
Fichiers .rst créés précédemment (sans extension)
.. codeauthor :: est une directive optionnelle

3 lignes complémentaires permettant de générer les pages d'index et la page de recherche.

5ème Etape : Compilation

- Génération de la documentation au format HTML
- Lancement de la commande >> **make html** à partir de la racine du dossier dans lequel se trouvent les programmes.





Résultat final

Accéder la documentation générée:

- Lancer le fichier « **index.html** »

Mise en oeuvre

Navigation (démo formateurs) sur les différents éléments générés:

- Pages d’accueil
- Pages de la table des matières (liens sources et docs)
- Modules
- Index
- Recherche

Aller plus loin.

Documentation complète (anglais)

http://www.sphinx-doc.org/en/master/index.html

Documentation libre (français) moins complète mais permettant une approche pas à pas

https://deusyss.developpez.com/tutoriels/Python/SphinxDoc

Le site readthedocs.org peut héberger la documentation d'un projet (en lien avec github)

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Gestions des fichiers structurés

Séquence 6

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Sommaire

- ❑ Le module pickle
- ❑ Le module CSV
- ❑ Le module JSON
- ❑ YAML et XML

Le module pickle: sérialisation

```
: import pickle
```

```
: mesobjets = [[1, 'r', 4], ('t', 'u', 'p'), {1: 'un' , 2: 'deux'}]
```

```
: with open('serial.dat' , 'wb')  as file:
    pickle.dump(mesobjets,file)
```

```
: mesobjets = []
```

```
: with  open('serial.dat', 'rb') as file:
    mesobjets = pickle.load(file)
print(mesobjets)
```

```
[[1, 'r', 4], ('t', 'u', 'p'), {1: 'un', 2: 'deux'}]
```

Ce module supporte aussi la compression des données

Le module CVS

manip_csv.ipynb

```
with open('igpde.csv', 'w') as fic:
    fic.write("'eric';'jeando';'martin'\n")
    fic.write("'infra';'mistral';'epn'\n")

import csv
with open('igpde.csv') as csvfile:
    lecteur = csv.reader(csvfile, delimiter=';')
    for row in lecteur:
        print(', '.join(row))
```

```
'eric', 'jeando', 'martin'
'infra', 'mistral', 'epn'
```

Le module JSON

Les fichiers JSON (JavaScript Object Notation) permettent de sérialiser des objets dans un format texte

→ lisible par les humains

→ modifiable avec un éditeur de texte

→ portable (indépendant des plate-formes)

→ standardisé (compatible avec d'autres langages)

==> Format très utilisé

YAML et XML

manip_xml.ipynb

Format yaml :

Installer le module pyyaml puis utiliser:

Les méthodes yaml.load pour charger un fichier et yaml.dump pour ecrire des données au format YAML.

Format XML :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<users>
    <user data-id="101">
        <nom>Zorro</nom>
        <metier>Danseur</metier>
    </user>
    <user data-id="102">
        <nom>Hulk</nom>
        <metier>Footballeur</metier>
    </user>
    <user data-id="103">
        <nom>Zidane</nom>
        <metier>Star</metier>
    </user>
    <user data-id="104">
        <nom>Beans</nom>
        <metier>Epicier</metier>
    </user>
    <user data-id="105">
        <nom>Batman</nom>
        <metier>Veterinaire</metier>
    </user>
    <user data-id="106">
        <nom>Spiderman</nom>
        <metier>Veterinaire</metier>
    </user>
</users>
```

```python
from lxml import etree

tree = etree.parse("data.xml")
for user in tree.xpath("/users/user/nom"):
    print(user.text)
```

```
Zorro
Hulk
Zidane
Beans
Batman
Spiderman
```

Voir aussi le parseur 100% Python:**defusedxml**

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

IHM

Séquence 7

Sommaire

- ❑ Tkinter
- ❑ Cinématique
- ❑ Exemple

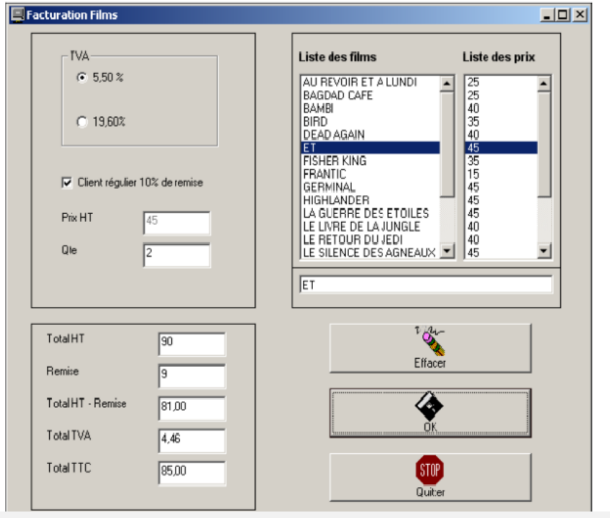
Client lourd avec Tkinter

Méthodes de placement:

- Grid (grille)
- Pack
- Place

Les composants:

- Boutons
- Labels
- Zones de saisie
- Boutons radio
- Listes
- etc.



Cinématique

- Une boucle d’exécution
- Une relation entre les composants à la gestion d’évènement

Soit directement dans la déclaration du composant

```
btn=Button(win, text="Press Enter ttk", command= callback)
```

Soit par un ‘bind’

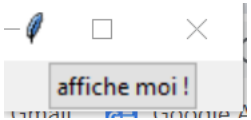
```
win.bind('<Return>',lambda event:callback())

win.mainloop()
```

```
def callback():
   Label(win, text="Hello World!", font=('Century 20 bold')).pack(pady=4)
```

Exemple

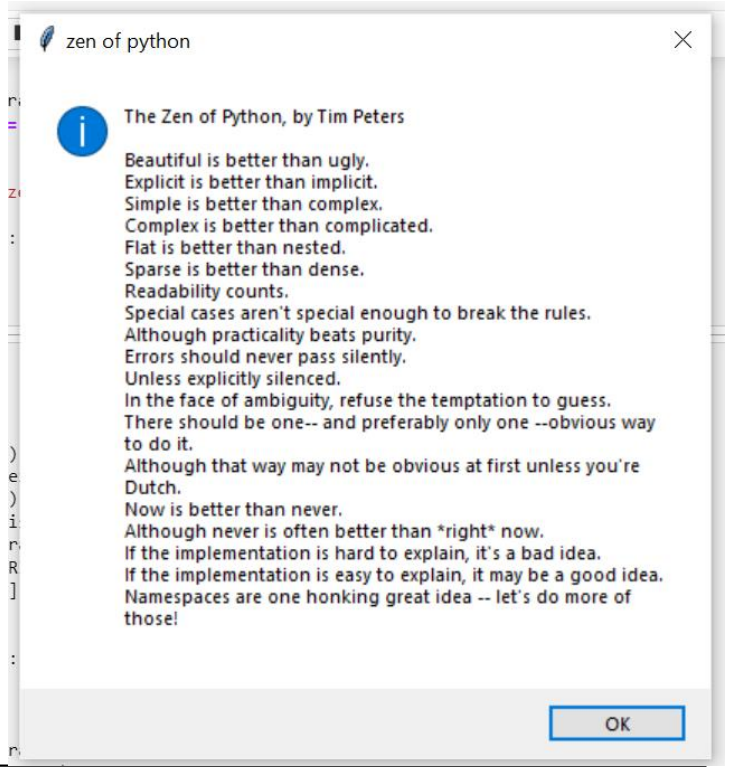
tp_zen.ipynb

Afficher le poème zen of Python (import this)



A partir de l'exemple (tp_zen)

- Remplacer le composant fenêtre par un composant texte muni d'une barre verticale de défilement (scrollbar)



Solution

1 à 3)

```
def show_zen():
    global frame2
    frame2.destroy()
    frame2 = Frame(frame)
    frame2.pack(side =TOP)
    text = Text(frame2, height=10)
    text.pack(side = LEFT)
    text.insert('1.0', this.s.translate(rot13) )
    scrollb = Scrollbar(frame2, command=text.yview)
    scrollb.pack( side = RIGHT, fill = Y )
    text['yscrollcommand'] = scrollb.set


if __name__ == "__main__":
    root = Tk()
    frame = Frame(root)
    frame.pack()
    zen_bouton = Button(frame, text ='affiche moi !', command= show_zen)
    zen_bouton.pack(side = TOP)
    frame2 = Frame(frame)
    root.mainloop()
```

4)

```
def show_zen():
    global frame2
    frame2.destroy()
    if zen_bouton['text']  == 'Effacer':
       zen_bouton['text']  = 'Afficher'
    else:
        frame2 = Frame(frame)
        frame2.pack(side =TOP)
        text = Text(frame2, height=10)
        text.pack(side = LEFT)
        text.insert('1.0', this.s.translate(rot13) )
        scrollb = Scrollbar(frame2, command=text.yview)
        scrollb.pack( side = RIGHT, fill = Y )
        text['yscrollcommand'] = scrollb.set
        zen_bouton['text'] = 'Effacer'
```

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

IGPDE

Conclusion

Les autres formations

Le module web (1 jour)

- Les bases de données avec SQLite et leurs manipulations avec Python
  - Manipulation SQL
  - Connexion avec un programme Python
- Réalisation d'une application WEB avec FLASK
  - Principes
  - Les routes et les méthodes
  - Les templates (jinja2)
- Butinage (scraping) du WEB avec beautifulSoup, Request et scrapy
  - Présentation de la chaine de récupération
  - Manipulations

La Datascience (1 jour)

- Numpy :manipulations de base
- Panda
  - Présentation
  - Manipulations
  - Nettoyage des données
- Visualisation des données
  - Matplotlib
- Quelques exemples de d’usages
  - Régression linéaire/logistique
- Les frameworks dédiés
  - Tensorflow/keras
  - Pytorch

RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

IGPDE

Python pour l'administration système sous Linux (1 jour)

- Dialoguer avec l'OS
  - Les fichiers
  - Les processus
- Le module sys
  - Utilisation des arguments
- Gestion des fichiers temporaires
- La gestion et la communication des processus

Annexe : Mise au point d'un programme

- Le module debug de python : **P**ython **D**e**B**ugger

Il s'utilise de deux façons

- En ligne de commande :

python -m**pdb** mon_programme.py

- Ou en prévoyant un appel dans le source

```
import pdb
print('je passe')
pdb.set_trace()
print('je stoppe')
```

```
je passe
--Return--
> <ipython-input-1-e7e068fa8cac>(3)<module>()->None
-> pdb.set_trace()

(Pdb)
```

Les commandes

? pour obtenir de l'aide.

```
(Pdb) ?

Documented commands (type help <topic>):
========================================
EOF     c          d        h         list      q        rv       undisplay
a       cl         debug    help      ll        quit     s        unt
alias   clear      disable  ignore    longlist  r        source   until
args    commands   display  interact  n         restart  step     up
b       condition  down     j         next      return   tbreak   w
break   cont       enable   jump      p         retval   u        whatis
bt      continue   exit     l         pp        run      unalias  where

Miscellaneous help topics:
==========================
exec  pdb

(Pdb)
```