



Programmer en Python

S'initier au langage

- Introduction à cette formation
 - Votre formateur ... Et Vous
 - Le matériel
 - Le support de cours
 - La distribution Anaconda
 - L'organisation – horaires
 - Formation de 4 jours
 - La forme :
 - Un mélange de concepts avec application directe par un exemple simple
 - Des exercices



Python

- Des liens utiles

- <https://www.python.org/> le site officiel
- <https://www.tutorialsteacher.com/python> *** très complet
- <https://www.w3schools.com/python/default.asp>
- <https://openclassrooms.com/fr/courses/7168871-apprenez-les-bases-du-langage-python>
cours OpenClassrooms
- <https://docs.python.org/3.6/tutorial/index.html> tutoriel
- <http://apprendre-python.com/> autre cours ***

- Encore un langage ! ?
- Installer Python
- Prise en main
- Les bases du langage
- Les fonctions
- Les modules
- Les exceptions
- Les chaînes de caractères
- Les listes, dictionnaires
- Les fichiers
- Mot de passe



- Classe, Objet, Attribut
 - Héritage
 - Polymorphisme
-
- Les outils pour la Data Science
 - Numpy
 - Mathplotlib
 - Pandas



- En option :
 - Interfaces graphiques :
 - wxPython
 - Glade
 - Multi-threading
 - Communication réseau
 - Générer un exécutable
 - Accéder à une base de données



A quoi sert Python ?

- Python :
 - Un langage puissant, riche et facile à apprendre
 - De nombreuses bibliothèques pour divers sujets
 - Manipulation de l'OS
 - Interaction avec des applications écrites dans un autre langage
 - ...
- Sous la forme :
 - De simples scripts pour des besoins particuliers
 - Des applications complètes, graphiques ou non
 - Des applications très complexes, distribuées sur plusieurs machines

- Utilisé pour configurer de nombreux équipements :
 - Serveurs de stockage en réseau – NAS –
 - Equipements de réseau – Cisco
 - Sur carte Raspberry (linux Raspbian)
 - Carte électronique MicroPython



- Mais encore :
 - Des bibliothèques d'outils pour la Data Science
 - Django, un **framework python open-source** consacré au développement web 2.0
 - Créer un serveur Web
 - Utilisation de SMTP (envoi de mails)
 - Websocket (communication asynchrone navigateur/serveur Web)

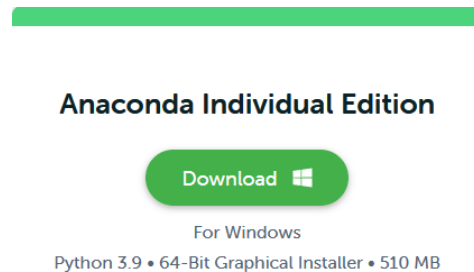
- Un langage interprété
 - Les instructions Python sont transcrites en langage machine au fur et à mesure de l'exécution
 - Pas de compilation
 - Portable : le même source compris sur Windows, Linux, iOS.
- Les versions
 - 1999 – 2000 versions 1.5 et 1.6
 - 2000 à 2010 versions 2.0 à 2.7
 - 2008 à 30/01/2018 versions 3.0 à 3.6
 - 31/01/2018 version 3.7
 - 12/2021 version 3.10.1

Le code des versions 2.x et 3.x n'est pas compatible
S'informer avant de choisir la version



<https://python.doctor/page-syntaxe-differente-python2-python3-python-differences>

- Nous allons utiliser la fourniture Anaconda dédiée à Python dans le contexte de la Data Science
- Aller sur le site
<https://www.anaconda.com/products/individual>
- Télécharger et installer



- Lancer Anaconda Navigator



- Lancer une fenêtre console par **Anaconda Prompt**
 - Taper **python -V**
 - Taper **python** le prompt **>>>** s'affiche
 - Réaliser quelques opérations :
 - $17 * 7$
 - $120 / 17$
 - $120 \% 17$

- Une « variable » sert à ranger/stocker une valeur (nombre, chaîne ...) lors du parcours du programme



- Une variable a un nom :
 - Ne doit pas commencer par un chiffre
 - Composé de lettres minuscules ou majuscules, chiffres, '_'
 - Sensible à la casse : nom, Nom, NOM sont des variables différentes

```
>>> nom='albert'  
>>> print(nom)  
albert
```

- Règle de nommage ?
 - Des recommandations sont faites dans PEP8 : <https://pep8.org/>

Dans ce cours on essayera :

- 1^{er} mot minuscule
- Autres mots, 1^{ère} lettre majuscule
- Ex : age, ageCourant, ageDepuisJanvier

- Type d'une variable :
 - Le « type » définit quel genre d'information est capable de contenir la variable : entier, chaîne, décimal ...
 - Il n'y a pas de déclaration de type en Python : il le fait au mieux :
 - `a=10` a contient un int ou un long en fonction de la taille
 - `a=3.14` a contient un float
 - `a="abcd"` a contient un str (string)
 - `a=5+4j` a contient un complex
 - `a=[1,5,6]` a contient une list
 -
 - La fonction `type(var)` fournit le type de la variable

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

- Dans la fenêtre console :

- `a=5`
- `a`
- `type(a)`
- `b=3.14`
- `type(b)`
- `a+b`
- `c="abcd"`
- `type(c)`
- `d=5+4j`
- `type(d)`



- Chaînes de caractères
 - A="chaîne de caractères" #Unicode par défaut en Python 3
 - A= 'chaîne de caractères'
 - Caractères spéciaux :
 - \" pour forcer un \"
 - \' pour forcer un \'
 - \n retour à la ligne
 - \t tabulation
 - \" \" \" pour chaîne sur plusieurs lignes et annuler les \" et \' intermédiaires
 - A= r"chaîne de caractères" pour annuler les caractères spéciaux
 - A= b"chaîne de caractères" pour chaîne en bytes

Les outils de développement, les IDE (integrated development environment), pour Python sont nombreux :

- Spyder
- Wing Personal
- Pycharm
- Visual Studio Code



- Créer un répertoire où nous allons créer le code des exercices et expériences
Répertoire « travailPython »
- Les outils de développement dédiés à des langages, permettant de gérer des projets, offrant un mode debug sont appelés des IDE (Integrated Development Environment)
- Les IDE pour Python sont nombreux : WingIDE, Spyder, PyCharm, Visual Studio Code , ... Nous choisissons Visual Studio Code



- Créer un répertoire où nous allons créer le code des exercices et expériences
Répertoire « travailPython »
- Lancer l'éditeur Spyder depuis Anaconda Navigator ou depuis la barre de recherche



– Nous allons éditer du code Python dans un fichier

- File -> New File
- Saisir le code

```
a=5  
print(a)
```

- Sauvegarder le fichier (essai1.py) et l'exécuter par F5

- Permet d'exécuter un bloc de code si une condition est réalisée

```
a=5  
print(a)  
if a > 3:  
    print("a supérieur à 3")
```

Mot clé if

test suivi de :

indentation

Opérateur	Signification littérale
<	Strictement inférieur à
>	Strictement supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

- else:

```
a=5
print(a)
if a > 3:
    print("a supérieur à 3")
else:
    print("a inférieur ou égal à 3")
```

- elif *test*:

```
a=2
print(a)
if a > 3:
    print("a supérieur à 3")
elif a>1:
    print("a supérieur à 1 et inférieur à 3")
else:
    print("a inférieur ou égal à 1")
```

- Mots clés and or not

```
a=2
print(a)
if a >= 1 and a <= 3:
    print("a compris entre 1 et 3")
else:
    print("a NON compris entre 1 et 3")
```

L'écriture

```
if 1 <= a <= 3 :
```

Est aussi permise

```
if not a == 3:
    print("a different de 3")

if a == 2 or a == 3 :
    print("a vaut 2 ou 3")
```

- Boucle for :

for element in collection:

instruction

instruction

element est une variable qui récupère chaque valeur de *collection*

collection est une suite de valeurs

Exemple de collections :

- Tab=[1,3,45,12] tableau appelé liste
- B=[1, "abcd", 123]
- Str="ceci est un tableau de caractères"
- C=range(10) liste de valeurs de 0 à 9
-

```
b=[1,2,45,4]
for element in b:
    print(element)

c=[1,"aze",45]
for elem in c:
    print(elem)

d="ceci est une chaîne"
for char in d:
    print(char)
```


- Boucle while

while condition:

instruction

instruction

```
nb = 7 # On garde la variable contenant le nombre dont on veut la table de multiplication
i = 0 # C'est notre variable compteur que nous allons incrémenter dans la boucle

while i < 10: # Tant que i est strictement inférieure à 10
    print(i + 1, "*", nb, "=", (i + 1) * nb)
    i += 1 # On incrémente i de 1 à chaque tour de boucle
```

```
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
4 * 7 = 28
5 * 7 = 35
6 * 7 = 42
7 * 7 = 49
8 * 7 = 56
9 * 7 = 63
10 * 7 = 70
```

- break
break stoppe une boucle

```
while 1: # 1 est toujours vrai -> boucle infinie
    lettre = input("Tapez 'Q' pour quitter : ")
    if lettre == "Q":
        print("Fin de la boucle")
        break
```

Préférer cette écriture :

```
lettre="a"
while lettre != "Q":
    lettre = input("Tapez 'Q' pour quitter : ")

print("Fin de la boucle")
```

- continue

continue force l'itération suivante de la boucle sans en sortir

```
i=1
while i in range(20): # equivalent à while i < 20
    if i%4==0:
        i += 1
        continue # on passe les valeurs multiples de 4
    print("i= ",i)
    i += 1
```

```
i= 1
i= 2
i= 3
i= 5
i= 6
i= 7
i= 9
i= 10
i= 11
i= 13
i= 14
i= 15
i= 17
i= 18
i= 19
...
```

- Utilisation de Wing IDE :
 - Debugger intégré
 - Aide en ligne
 - Aide à la saisie

Télécharger et installer l'outil, version Personal

<http://www.wingware.com/downloads/wing-personal/9.1.2.1/binaries>

Windows Installer
32-bit and 64-bit

- Exercice 1



- Une fonction est un bloc d'instructions ré utilisable
- Évite d'écrire plusieurs fois le même code
- Rend un programme « modulaire »

```
def nomFonction(parametre1, parametre2, ...):  
    #bloc d'instructions  
    return valeur    # optionnel
```

```
def tableMultiplication(nb, max):  
    i = 0  
    while i < max: # Tant que i est strictement inférieur à la variable max,  
        print(i + 1, "★", nb, "=", (i + 1) ★ nb)  
        i += 1  
  
tableMultiplication(7,10)  
tableMultiplication(9,12)
```

```
1 ★ 7 = 7  
2 ★ 7 = 14  
3 ★ 7 = 21  
4 ★ 7 = 28  
5 ★ 7 = 35  
6 ★ 7 = 42  
7 ★ 7 = 49  
8 ★ 7 = 56  
9 ★ 7 = 63  
10 ★ 7 = 70
```

- Valeurs par défaut

```
def tableMultiplication(nb, max=10):  
    i = 0  
    while i < max: # Tant que i est strictement inférieur à la variable max,  
        print(i + 1, "*", nb, "=", (i + 1) * nb)  
        i += 1  
  
tableMultiplication(7,10)  
tableMultiplication(9,12)  
tableMultiplication(6)           # equivalent à tableMultiplication(6,10)  
tableMultiplication(max=8,nb=4)  # nommage explicite des paramètres
```

return

- Permet à la fonction de renvoyer une valeur au programme appelant

```
def carre(val):  
    calcul = val * val  
    return calcul          # return val * val    est plus direct  
  
maValeur=carre(15)  
print(carre(10))  
print(carre(12))
```

- Plusieurs valeurs peuvent être retournées

```
def decomposer(entier, divisePar):  
    """Cette fonction retourne la partie entière et le reste de  
    entier / divisePar"""  
  
    partieEntiere = entier // divisePar  
    reste = entier % divisePar  
    return partieEntiere, reste          # retourne 2 valeurs  
  
a, b = 7 , 13.5                        # affecte a avec 7 et b avec 13.5  
partie, leReste = decomposer(13, 3)    # les 2 valeurs de restuen sont rangées dans  
                                         # partie et leReste
```

- Il est possible d'ajouter à la définition d'une fonction le type des paramètres et le type de la valeur de retour

```
def trait( age : int, prenom : str) -> str :  
    ch = prenom + ", vous avez " + str(age) + " ans"  
    return ch
```

```
print(trait("15","Jerome") )
```

```
Jerome, vous avez 15 ans
```

ces annotations ne sont utilisées qu'à titre informatif (doc on line),
l'interpréteur ne les utilise pas

- Une autre façon de déclarer des fonctions très simples
var = **lambda** arg1, arg2, ... : instruction de calcul de retour

```
f = lambda x, y : x + 5 * y  
  
print(f(10,0))  
print(f(1,2))
```

- Un module est un ensemble de fonctions relatives à un même sujet
- Il suffit d'importer un module pour bénéficier du code des fonctions
- Deux méthodes :

- `import module`

```
import math  
print(math.cos(math.pi))
```

- `from module import fonction`

- `from module import *`

```
from math import cos, pi  
print(cos(pi))
```

```
from math import *  
print(cos(pi))
```

- Nous allons créer notre propre module : exercice 2



- Au dessus de la notion du module existe la notion de « package » : un regroupement de modules
Pas compliqué, exercice 2 bis .



- Tous les modules Python ne sont pas installés sur la machine
- <https://pypi.org/> permet la recherche d'un module/package
- <http://apprendre-python.com/page-pip-installer-librairies-automatiquement>
pour une explication de PIP
- Pour installer un module, dans une fenêtre commande, utiliser
`pip install module`
- Ou dans le navigateur Anaconda, utiliser l'onglet Environments

- Pour installer pip (inutile avec Anaconda):

Sur le site <https://pip.pypa.io/en/stable/installation/> récupérer le fichier get-pip.py

- Dans une fenêtre commande exécuter
`python get-pip.py`

- Regarder si matplotlib est déjà installé. Si non, l'installer :

- Pour installer matplotlib :

- `pip install matplotlib`
- `pip list` pour liste des modules installés



- Quand Python rencontre une erreur dans le code, il lève une exception pour informer de cette erreur.

```
Traceback (most recent call last):  
  File "C:/Users/Daniel/_Projets/Cours(  
    99, in <module>  
      val = 5 / 0  
ZeroDivisionError: division by zero  
...
```

- C'est au programme de traiter cette information

```
try:  
    val = 5 / 0  
except :  
    print("Exception levée")
```

```
try:  
    val = 5 / 0  
except Exception as exception_retournee:  
    print("Voici l'erreur :", exception_retournee)
```

```
try:
    # Test d'instruction(s)
except TypeError:
    # Traitement en cas d'erreur
finally:
    # Instruction(s) exécutée(s) qu'il y ait eu des erreurs ou non
```

- Test avec **assert**
 - Permet de lever l'exception `AssertionError` sur résultat de test `False`

```
annee = input("Saisissez une année supérieure à 0 :")
try:
    annee = int(annee) # Conversion de l'année
    assert annee > 0
except ValueError:
    print("Vous n'avez pas saisi un nombre.")
except AssertionError:
    print("L'année saisie est inférieure ou égale à 0.")
```

- Lever une exception : **raise** (voir w3schools)
- Exercice 3



- Une chaîne de caractères est de classe str
- Bénéficie donc de toutes les méthodes de cette classe
 - `help(str)` pour obtenir la liste

https://www.w3schools.com/python/python_strings.asp

```
print("AbCd".lower())
```

- Formatage d'une chaîne : `.format`

```
prenom="John"
nom="Smith"
rue="6 rue du Bas"
Ville="Marseille"

print("Mr {0} {1} , adresse {2} , ville de {3}. Bonjour Mr {1}".format(prenom,nom,rue,Ville))

# syntaxe avec nommage des paramètres
print("Mr {pr} {nom} , adresse {ru} , ville de {vil}. Bonjour Mr {nom}".\
      format(nom=nom,pr=prenom,ru=rue,vil=Ville))
```

```
Mr John Smith , adresse 6 rue du Bas , ville de Marseille. Bonjour Mr Smith
```


Autre façon de formater une chaîne

```
prenom="John"  
nom="Smith"  
rue="6 rue du bas"  
ville="Marseille"  
taille=172.235
```

```
phrase = f"Mr {prenom} {nom}, habite {rue} à {ville} et mesure {taille:.2f} cm"  
print(phrase)
```

Affiche **Mr John Smith, habite 6 rue du bas à Marseille et mesure 172.24 cm**

- Concaténation de chaînes : +

```
val = "Bonjour " + "tout le monde"
print(val)

temp=37
val = "la température est de " + str(temp) + " degrés"
print(val)
```

- Caractères dans une chaîne

```
chaine="Bonjour à tous"
print(len(chaine))      # taille de la chaine      14
print(chaine[0])        # 1er char                B
print(chaine[-1])       # dernier char            s

print(chaine[0:3])      # 3 premiers char         Bon
print(chaine[:3])       # idem                    Bon
print(chaine[2:len(chaine)]) # du 3eme au dernier char  njour à tous
print(chaine[2:])       # idem                    njour à tous
                                                    \home\user\perf

chaine = "/home/user/perf"
chaine2=chaine.replace("/", "\\")
print(chaine2)
```

- Une chaîne de caractères est un conteneur de char
- Une liste est un conteneur d'objets

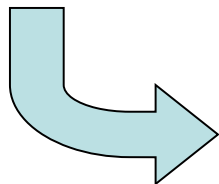


```
uneListe=list()      #liste vide
print(type(uneListe))
uneListe=[]          #idem

autreListe = [1, 2, 3, 4, 5]      # Une liste avec cinq objets
autreListe = [1, "abcd", 3.25, 4, 5]
autreListe[1] = 10                # "abcd" remplacé par 10
autreListe.append("Hello")        # "Hello" ajouté en fin de liste
print(autreListe)
autreListe.insert(2, "inséré")     # insère à l'index 2
print(autreListe)

del autreListe[0]                  # supprime l'élément 0
print(autreListe)

autreListe.remove("inséré")        # supprime l'élément indiqué par sa valeur
print(autreListe)
print("contenu de la liste : " + str(autreListe))  # print("contenu de la liste : " + autreListe)
```



```
<class 'list'>
[1, 10, 3.25, 4, 5, 'Hello']
[1, 10, 'inséré', 3.25, 4, 5, 'Hello']
[10, 'inséré', 3.25, 4, 5, 'Hello']
[10, 3.25, 4, 5, 'Hello']
contenu de la liste : [10, 3.25, 4, 5, 'Hello']
.
```

- Parcours des listes

```
maListe = [1, "abcd", 3.25, 4, 5]
```

```
i=0
while i < len(maListe):
    print(maListe[i])
    i+=1
```

```
# meilleure écriture
print("utilisation de for")
for elem in maListe:
    print(elem)
```

```
1
abcd
3.25
4
5
utilisation de for
1
abcd
3.25
4
5
```

- enumerate(liste)

```
for elem in enumerate(maListe):
    print(elem)
```

```
for index, elem in enumerate(maListe):
    print("À l'indice {0} se trouve {1}.".format(index, elem))
```

```
(0, 1)
(1, 'abcd')
(2, 3.25)
(3, 4)
(4, 5)
À l'indice 0 se trouve 1.
À l'indice 1 se trouve abcd.
À l'indice 2 se trouve 3.25.
À l'indice 3 se trouve 4.
À l'indice 4 se trouve 5.
```

- Exercice 4



- tuple : N-uplet en français
- Un tuple est une liste qui ne peut plus être modifiée après sa création

```
monTuple= (1, "ok", "Apprenez Python")
print(type(monTuple))
print(monTuple[2])          # même syntaxe que pour une liste
#monTuple[0] = 3            # interdit
```

for elem in monTuple:
 print(elem)

<class 'tuple'>
Apprenez Python
1
ok
Apprenez Python

- Utilisation sur return de fonction

```
def donneMoiTonNom():  
    return ("olivier", "engel")
```

prenom, nom = donneMoiTonNom()
print("Mr {0} {1}".format(prenom, nom))

prenom = donneMoiTonNom()[0]
print(prenom)

Mr olivier engel
olivier

- Transformer une chaîne en liste : `split()`

```
chaine="Qui n'a pas signé la feuille?"
resultat = chaine.split(" ")          # l'espace est le critère de découpage
print(resultat)
```

```
path = "c:\\program files\\Data"
resultat = path.split("\\")          # \ est le critère de découpage
print(resultat)
```

```
['Qui', 'n'a', 'pas', 'signé', 'la', 'feuille?']
['c:', 'program files', 'Data']
```

- Transformer une liste en chaîne: `join()`

```
path = "c:\\program files\\Data"
resultat = path.split("\\")          # \ est le critère de découpage
print(resultat)
```

```
assemble="/".join(resultat)          # chaque element est assemblé avec '/'
print(assemble)
```

```
['c:', 'program files', 'Data']
c:/program files/Data
```

- La « compréhension » permet de filtrer et modifier les listes

```
liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
listeCarree = [nb * nb for nb in liste]
print(listeCarree)

listePaire = [nb for nb in liste if nb % 2 == 0]
print(listePaire)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[0, 2, 4, 6, 8, 10]
```

- Un dictionnaire est une sorte de liste contenant des couples (clé, valeur)

```
dico={} # dictionnaire vide
dico=dict() # equivalent
dico["Dupont"] = "6 rue des tilleuls"
dico["Wilson"] = "24 bd Albert"

print(dico)
dico["Dupont"] = "56 bd Tourelle" # modification

etal={"pomme": 30, "poire": 40, "orange" : 10}
print(etal["pomme"])
print(etal.get("pomme"))

for cle in etal.keys():
    print(cle)
for cle in etal.values():
    print(cle)

{'Dupont': '6 rue des tilleuls', 'Wilson': '24 bd Albert'}
30
30
pomme
poire
orange
30
40
10
```


- OPTION :

Un peu de détente : réaliser les exercices du fichier
ExoTurtle.pdf



- Chemins absolu et relatif vers un fichier
 - Un chemin absolu est une chaîne qui décrit tous les répertoires depuis la racine jusqu'au fichier. Le '/' est le séparateur. Sous Windows '\' est permis mais l'on préfère '/'
ex `C:/Users/AppData/appli/fichier.txt`
 - Un chemin relatif permet d'indiquer un endroit au dessus ../ ou en dessous dans les répertoires
ex `../../Data/fichier.txt`
- Ouvrir un fichier : `open()` avec le mode d'ouverture
 - r**, pour une ouverture en lecture (READ).
 - w**, pour une ouverture en écriture (WRITE), à chaque ouverture le contenu du fichier est écrasé. Si le fichier n'existe pas python le crée.
 - a**, pour une ouverture en mode ajout à la fin du fichier (APPEND). Si le fichier n'existe pas python le crée.
 - b**, pour une ouverture en mode binaire.
 - t**, pour une ouverture en mode texte.
 - x**, crée un nouveau fichier et l'ouvre pour écriture

- Fermer un fichier : méthode `close()`
- Lire le contenu entier : méthode `read()`
- Ecrire le contenu : méthode `write()`

```
fichier = open("fichier.txt", 'r')
contenu = fichier.read()
print(contenu)
fichier.close()
```

```
fichier = open("fichier1.txt", 'a')
fichier.write("\nNouvelle ligne")
fichier.close()
```

- Afin d'éviter d'utiliser – et d'oublier – le `close()` :
utiliser `with`

```
with open("fichier1.txt", 'r') as file:  
    print(file.read())
```



- Exercice 5

- Utilisation du module « os » , os.path

- help("os.path")

exercice 6



<code>abspath(path)</code>	→ Retourne un chemin absolu
<code>basename(p)</code>	→ Retourne le dernier élément d'un chemin
<code>commonprefix(list)</code>	→ Retourne le chemin commun le plus long d'une liste de chemins
<code>dirname(p)</code>	→ Retourne le dossier parent de l'élément
<code>exists(path)</code>	→ Test si un chemin existe
<code>getaTime(filename)</code>	→ Retourne la date du dernier accès au fichier [<code>os.stat()</code>]
<code>getctime(filename)</code>	→ Retourne la date du dernier changement de métadonnées du fichier
<code>getmtime(filename)</code>	→ Retourne la date de la dernière modification du fichier
<code>getsize(filename)</code>	→ Retourne la taille d'un fichier (en octets)
<code>isabs(s)</code>	→ Test si un chemin est absolu
<code>isdir(s)</code>	→ Test si le chemin est un dossier
<code>isfile(path)</code>	→ Test si le chemin est un fichier régulier
<code>islink(path)</code>	→ Test si le chemin est un lien symbolique
<code>ismount(path)</code>	→ Test si le chemin est un point de montage
<code>join(path, s)</code>	→ Ajoute un élément au chemin passé en paramètre
<code>normcase(s)</code>	→ Normalise la casse d'un chemin
<code>normpath(path)</code>	→ Normalise le chemin, élimine les doubles barres obliques, etc.
<code>realpath(filename)</code>	→ Retourne le chemin canonique du nom de fichier spécifié (élimine les liens s
<code>samefile(f1, f2)</code>	→ Test si deux chemins font référence au même fichier réel
<code>sameopenfile(f1, f2)</code>	→ Test si deux objets de fichiers ouverts font référence au même fichier
<code>split(p)</code>	→ Fractionne un chemin d'accès. Retourne un tuple

- Le module pickle permet d'enregistrer et de relire des objets dans un fichier binaire

```
# pickle
import pickle
scoreDico = {"joueur 1" : 4, "joueur 2" : 30, "joueur 3" : 109 }
tabVille = ["Paris", "Nantes", "Bordeaux"]

with open("dataPickle", 'wb') as fichier:
    monPickle = pickle.Pickler(fichier)
    monPickle.dump(scoreDico)
    monPickle.dump(tabVille)

# récupération unpickle

with open("dataPickle", 'rb') as fichierRecup:
    monDepickle = pickle.Unpickler(fichierRecup)
    dico = monDepickle.load()
    tab = monDepickle.load()
print(dico)                {'joueur 1': 4, 'joueur 2': 30, 'joueur 3': 109}
print(tab)                 ['Paris', 'Nantes', 'Bordeaux']
```

- « portée » signifie où les variables sont elles accessibles?

```
a=5

def fct():
    print("La variable a contient {}".format(a))
    #a=12          provoque une exception a en lecture seule

fct()
a=10
fct()

# utilisation de 'global'
b=4
def fct():
    global b      # permet que b soit accessible partout
    print("La variable b contient {}".format(b))
    b=12

fct()
print(b)
```

La variable a contient 5
La variable a contient 10

La variable b contient 4
12

- La bonne écriture d'une fonction est qu'elle consomme des paramètres d'entrée, utilise des variables locales, retourne la/les valeurs : pas de variables externes à la fonction

```
# bon usage

def fct(param) :
    print("Le param d'entrée contient {}".format(param))
    return param + 1

val = fct(4)
```


- Permet de trouver/vérifier si une chaîne a une forme attendue.

Ex : forme d'une adresse IP, adr mail, site web, no tel ...

- Aller sur le site :

<http://apprendre-python.com/page-expressions-regulieres-regular-python>

```
import re          # module regular expression

print(re.match(r"GR(.)?S", "GRIS") != None)      # True
print(re.match(r"^A(.)+Z$", "ABCDZ") != None)    # True

print(re.match(r"^[a-zA-Z0-9_+]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$)", "fred.albert@free.fr") != None)

s = "dans cette chaîne existe une adresse ip 18.18.186.56 comme celle ci"
liste = re.findall(r'\b25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?\.\b25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?\.\b25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?\.\b25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?\b', s)
print(liste)

True
True
True
['18', '18', '186', '56']
```

- Le module getpass gère le chiffrement

```
import hashlib
from getpass import getpass

chaine_mot_de_passe = b"azerty"
mot_de_passe_chiffre = hashlib.sha1(chaine_mot_de_passe).hexdigest()

verrouille = True
while verrouille:
    entre = getpass("Tapez le mot de passe : ") # azerty
    # On encode la saisie pour avoir un type bytes
    entre = entre.encode()

    entre_chiffre = hashlib.sha1(entre).hexdigest()
    if entre_chiffre == mot_de_passe_chiffre:
        verrouille = False
    else:
        print("Mot de passe incorrect")

print("Mot de passe accepté...")
```

- Les scripts sources peuvent être commentés pour aider les développeurs : écrire des docstrings:
 - Chaînes de texte juste en dessous de fonctions ou de classes
 - Chaîne tout en haut d'un module

```
$ python
>>> class FooBar(object):
...     "ma doc de classe"
...
...     def une_methode():
...         "ma doc de methode"
...         print 'boo'
...
```

- Le Vintage Téléphone et le e-Phone



C'est à l'appelant :

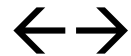
- de connaître le n°
 - de consulter un annuaire
 - de composer le n°
- ➔ Aucune donnée
- ➔ Une seule fonction



C'est l'Objet téléphone

- qui possède les n°
 - qui réalise l'appel à partir d'un nom
 - il est facile d'ajouter une fonction/application
 -
- ➔ Contient ses données propres
- ➔ Les fonctions utilisent leurs données propres

- La programmation Procédurale



VS la programmation Objet

Procédural :

- écriture d'un code principal
- écriture de fonctions
- C'est au programmeur d'orchestrer l'appel de l'ensemble des fonctions en fournissant les bonnes données
- ➔ nécessite grande maîtrise
- ➔ Nécessite des variables globales
- ➔ évolutions coûteuses

Objet :

- un objet détient ses données et ses fonctions pour traiter ses données
- protège ses données
- Le programme principal est réduit
- Le programmeur appelle des fonctions de l'objet
- ➔ travail modulaire possible
- ➔ évolutions faciles

- L'approche Objet

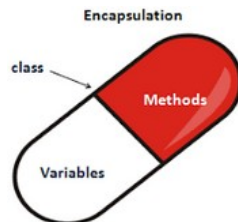
Imaginer des « boîtes » autonomes qui contiennent leurs données et tout ce qu'il faut en interne pour les manipuler

Il faut donc imaginer ce que contient la boîte (les données) et ce que cette boîte doit faire avec (les fonctions ou méthodes).

Ce raisonnement se nomme l' « **Abstraction** ».



Le fait de mettre en boîte et de cacher/montrer ce qu'il faut et ce qu'il ne faut pas nomme l' « **Encapsulation** »



- Encapsulation

- Consiste à mettre dans une même boîte le code (Méthodes) et les données (les Attributs)
- Consiste à masquer les Attributs et Méthodes qu'il est dangereux/inutile de montrer
- Consiste à montrer aux autres objets le juste suffisant.

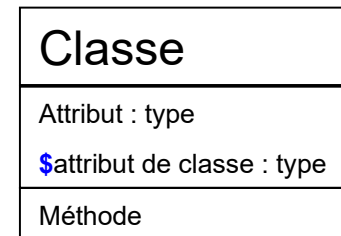
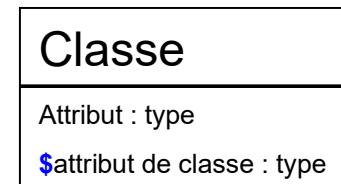
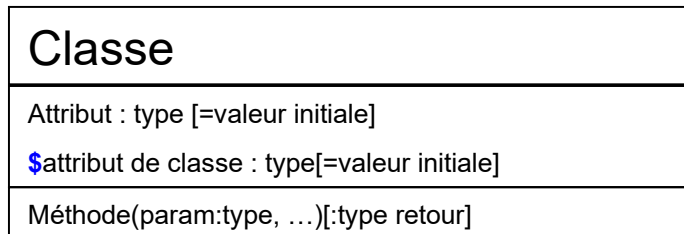


- Comment définir la visibilité/masquage :

- Python ne dispose pas complètement de ce mécanisme !
- Tout attribut ou méthode est accessible par les autres classes
Équivaut au mot clé « public » dans d'autres langages

- Formalisme graphique en UML. Plusieurs niveaux possibles :

- Classe
- Classe + Attributs
- Classe + Attributs + Méthodes
- Diagramme complet



Visibilité

- Membre privé
- + Membre public
- # membre protégé

- Exemple

Rectangle
<ul style="list-style-type: none">- origine : Point- hauteur : int- largeur : int
<ul style="list-style-type: none">+ Rectangle(origine:Point,larg:int, haut:int)+ Perimetre() : float+ Translater(point:Point)+ Afficher()

- Utilité des Classes
 - Une Classe est un « moule » à Objets
 - Elle définit :
 - Les données, appelés attributs (mais ne les possède pas)
 - Les traitements sur ses données : les méthodes
 - Le diagramme de classe de la méthode UML est à utiliser en conception

- Définition d'une classe

`class Point:`

 "Définition d'un point "

 # code indenté

- Définition des attributs

Non formelle comme d'autres langages : c'est la 1ère utilisation qui déclare un attribut, avec le mot « `self` »

- Constructeur : méthode appelée à la création de l'objet

Non obligatoire

```
def __init__(self, param ..) :  
# affectation des données
```

- Méthode de classe :

```
def maMethode(self, param ..) :  
# affectation des données et traitements
```

- Créer un objet :

```
obj = nomClasse(param...)
```

- Supprimer un objet :

```
del obj
```

- Exemple :

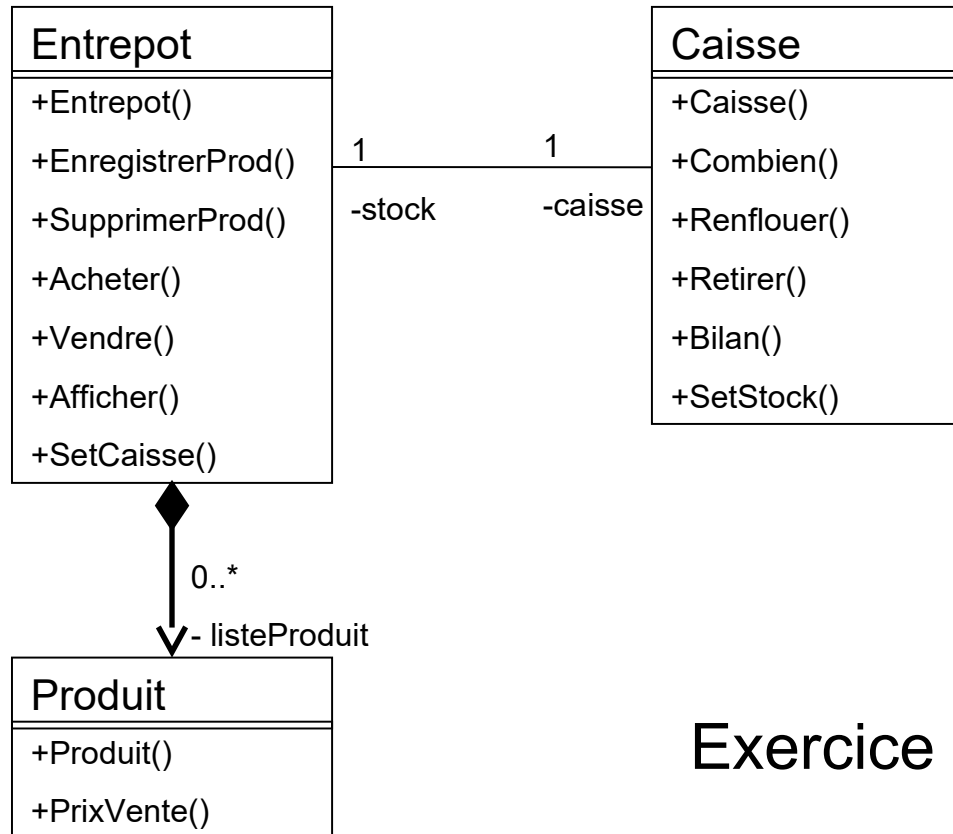
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("John", 36) # p1 est un objet
p1.myfunc()
```

- Dans une application Objet les Objets interagissent entre eux, ils sont en **relation**
Les diagrammes UML permettent de montrer ces relations :

Exemple



Exercice 8

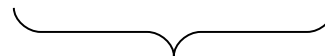


- Dans la conception Objet il existe la notion de classe mère et classe fille
- Répond au critère « est une sorte de »
Ex : **Voiture** est une sorte de **Véhicule**
Banane est une sorte de **Fruit**
Cercle est une sorte de **Forme**
VoitureDeSport est une sorte de **Voiture** ...



Cas particulier

Classe enfant



Cas général

Classe parent

- L'intérêt est alors de Factoriser le développement dans le code de la classe parent
- Et de coder les différences dans la/les classes enfant
- `class` NomClasseEnfant(NomClasseParent) :
- Il est alors possible de redéfinir, surcharger, une méthode existante dans la classe enfant
- Dans la classe enfant il est possible d'appeler une méthode de la classe parent :
NomClasseParent.methode()
- Exercice 9



- Soit un lot d'objets de classes différentes mais issues d'une même classe mère
- Le polymorphisme de classe permet à partir d'une référence d'un objet, d'appeler la méthode correspondant correctement à sa classe

- Ex :

```
p1 = Voiture("C3", 2.3,2.5)
```

```
p2 = VoitureDeSport ("Ferrari", 2.3,2.5,250)
```

```
p3=p1
```

```
p3.Presenter()          # appelle Voiture.Presenter()
```

```
p3=p2
```

```
p3.Presenter()          # appelle
```

```
VoitureDeSport.Presenter()
```

Exercice 10



Les fonctions magiques ou Dunder

- Dunder signifie Double Underscore : `__xx__`
- Les Dunders ont diverses fonctions, en particulier la surcharge des opérateurs
- Ils interviennent dans la définition des classes

<https://docs.python.org/3/reference/datamodel.html>

Il est ainsi possible de surcharger :

Les opérateurs de comparaison

Les opérateurs arithmétiques simples (+, -, * ...) et combinés (+=, -=, ...)

Surcharge de l'addition et soustraction pour la classe Point

```
class Point:
```

```
    def __init__(self, x:int, y:int)-> None:
```

```
        self.x = x
```

```
        self.y = y
```

```
    def __add__(self, other:Point):
```

```
        return Point(self.x + other.x, self.y + other.y)
```

```
    def __sub__(self, other:Point):
```

```
        return NotImplemented
```

```
    def __str__(self) -> str:
```

```
        return "point __str__ " + str(self.x) + " " + str(self.y)
```

```
P1 = Point(10,10)
```

```
P2 = Point(20,20)
```

```
P3 = P1 + P2
```

Une fonction ou méthode qui utilise l'instruction `yield` est une fonction Generator

Une telle fonction retourne quand elle est appelée un objet `iterator`, objet qui doit être utilisé pour exécuter le corps de la fonction Generator

`iterator.__next__()` provoque l'exécution du corps du Generator

Regarder et exécuter l'exemple fourni `generator.py`

Le détail sur <https://python.developpez.com/tutoriels/apprendre-programmation-fonctionnelle/>

C'est un style de programmation qui utilise comme paramètres des fonctions de traitement.

Voici deux exemples :

```
name_lengths = map(len, ["Mary", "Isla", "Sam"])
Print(list(name_lengths))
# => [4, 4, 3]
```

```
squares = map(lambda x: x * x, [0, 1, 2, 3, 4])
print(list(squares))
# => [0, 1, 4, 9, 16]
```

Python Syntaxe avancée : Décorateurs

Ce terme regroupe 2 notions distinctes en Python :

- le design pattern qui permet d'ajouter des fonctionnalités à une fonction
- des annotations (terme Java) pour augmenter la définition d'une classe

Analyser le fichier Decorateurs.py

Un context manager, gestionnaire de contexte est utile au programmeur lorsqu'il y a accès à des ressources que l'on doit à terme libérer, en fin d'utilisation, ou sur apparition d'une exception

Par exemple pour l'accès à un fichier :

```
f = open('log.txt', 'w')
```

```
try:
```

```
    f.write('hello world')
```

```
except Exception:
```

```
    pass
```

```
finally:
```

```
    f.close()
```

Préférer l'écriture :

```
with open('log.txt', 'w') as f:
```

```
    f.write('hello world')
```


Python Syntaxe avancée : with et ContextLib

pour que l'instruction with fonctionne sur l'un de nos développements, il faut implémenter un Context manager

Un contextManager est une classe qui implémente les méthodes

- `__enter__` pour allouer la ressource
- `__exit__` pour libérer la ressource

Utiliser Contextlib.py

Pour réaliser des tests unitaires, python dispose du module unittest

<https://docs.python.org/3/library/unittest.html>

<https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en-python/2235416-creez-des-tests-unitaires-avec-unittest>

Exemple :

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

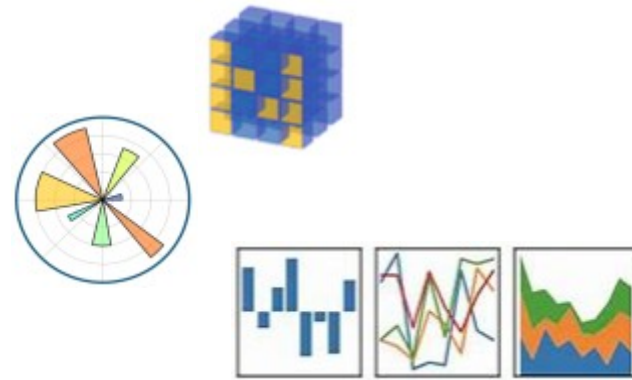
    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

- Les outils pour la Data Science

- Numpy
- Mathplotlib
- Pandas



- Le site

<https://openclassrooms.com/fr/courses/4452741-decouvrez-les-librairies-python-pour-la-data-science>

peut être très utile

- Nous allons d'abord nous familiariser avec Jupyter Notebook, un atelier de calcul pour les Data Scientists
- Lancer Jupyter Notebook et charger le fichier Presentation.ipynb



- Bibliothèque logicielle libre pour créer des matrices et tableaux multidimensionnels
- Offre des fonctions pour les manipuler
- Conçu pour être performant en exécution
- Le Memento joint au cours fournit un résumé des commandes

<https://courspython.com/apprendre-numpy.html>

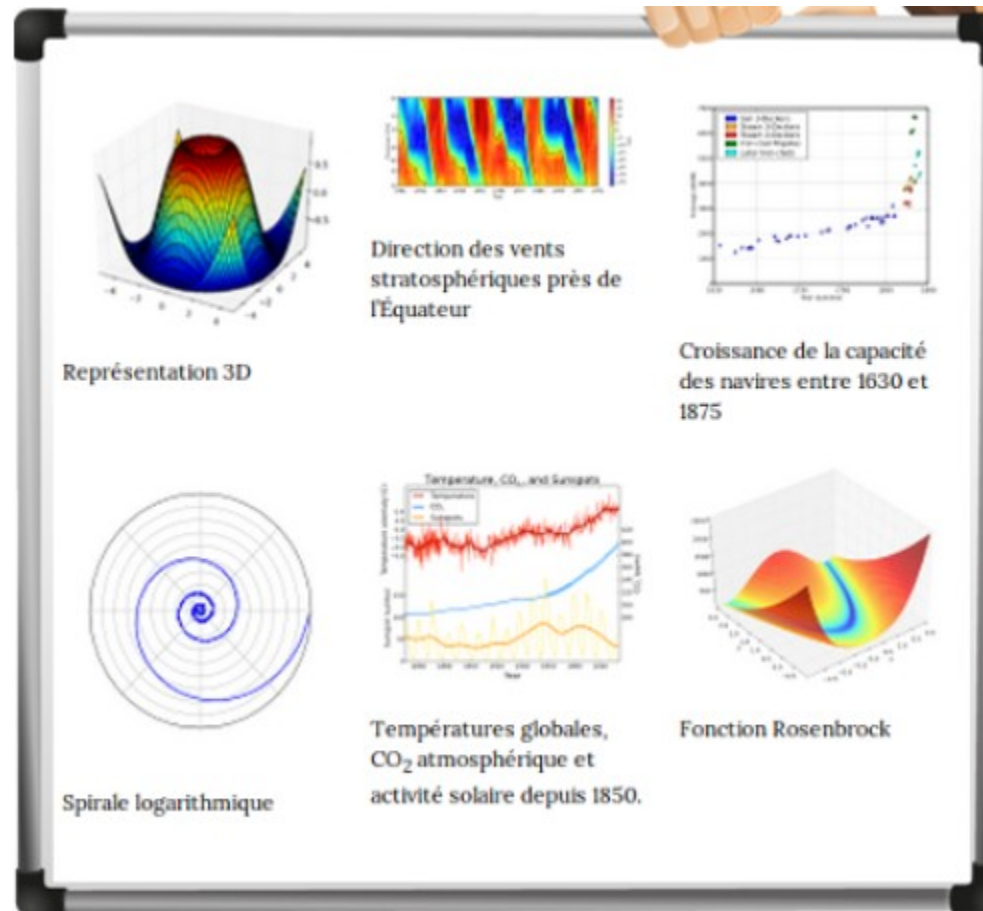
<https://numpy.org/doc/stable/>

- Quelques manipulations : Exercice 16



Python Représentation graphique : Matplotlib

- <http://apprendre-python.com/page-creeer-graphiques-scientifiques-python-apprendre>



Exercice



- Bibliothèque logicielle libre complémentaire à numpy pour manipuler et analyser les données
- Gère la notion de DataFrame : conteneur de np.array pour les manipuler
- DataFrame : 2 dimensions
- Panel pour 3 dimensions ou plus
- Permet d'appliquer des traitements d'algèbre relationnelle
- A découvrir sur <https://openclassrooms.com/fr/courses/4452741-decouvrez-les-librairies-python-pour-la-data-science/5558996-passez-de-numpy-a-pandas>

- Le machine Learning est une branche de l'intelligence artificielle.
- Permet à un logiciel d' « apprendre » par lui même en « observant » un lot important de données
- En fonction des données d'expérimentation que prendra l'algorithme d'apprentissage en entrée, il déduira par lui même une hypothèse de fonctionnement. Il utilisera cette dernière pour de nouveaux cas, et affinera son expérience au fil du temps.

- On distingue deux catégories de ML :
 - Apprentissage supervisé
On fournit un jeu de données d'apprentissage (training set) avec des variables prédictives X , et Y le résultat de l'observation.
L'algorithme va trouver une fonction qui permet de terminer au mieux Y en fonction de X
Ex : Prix d'une maison en fonction de multiples critères
 - Apprentissage non supervisé
A partir d'un lot de données le programme essaie de trouver des structures par lui même, des regroupements de données (clustering).

- Dans le domaine du machine Learning :
 - SciKit-Learn pour l'apprentissage automatique
 - TensorFlow, open source développé par Google d'apprentissage automatique
 - PyTorch open source sur une base développée par Facebook d'apprentissage automatique

- Plusieurs bibliothèques existent pour créer des interfaces graphiques homme/machine (IHM) :
 - GTK
 - TKinter
 - wxPython
 -
- wxPython est choisi ici car
 - Multi-plateforme Unix, Mac, Windows
 - Look récent
 - Uniforme avec wxWindows du C++
 - Il existe un éditeur de de dessin : wxGlade

Python Représentation graphique : wxPython

- <https://wxpython.org/pages/overview/>
- Documentation <https://docs.wxpython.org/>



- Exercice 11
- Les points principaux :
 - Import wx
 - class HelloFrame(wx.Frame):
 - self.Bind(wx.EVT_MENU, self.OnHello, helloItem)

Python Représentation graphique : wxPython

- Créer à la main du code wxPython est fastidieux et risqué :
Nous allons utiliser un éditeur wysiwyg : Glade
- Installer l'éditeur Glade :
<https://sourceforge.net/projects/wxglade/files/wxglade/0.8.0/>
- Exercice 12



- <https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en-python/2235545-la-programmation-parallele-avec-threading>
- Le multi threading permet l'exécution en parallèle de plusieurs portions de code
- Un thread est une unité d'exécution de code
- Charger et exécuter les fichiers thread1.py à thread4.py

A retenir : classe `Thread`, méthodes `__init__` `run` `start` `join`

classe `RLock`

- <https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en-python/234698-le-reseau>
- <http://apprendre-python.com/page-reseaux-sockets-python-port>
- Regarder les fichiers SocketServer.py et SocketClient.py

- Il est possible de distribuer une application Python sur des postes où Python n'est pas installé :
cx_Freeze est un outil qui génère un exécutable et satellites juste nécessaires.
Compatible Python 2.x et 3.x
Multi plateformes.
- Installation par
`pip install cx_Freeze --upgrade`

- Créer le fichier setup.py (nom libre)

```
from cx_Freeze import setup, Executable

# On appelle la fonction setup
# remplacer exoTreeView par le nom de votre script
setup(
    name = "exoTreeView",
    version = "1",
    description = "Votre programme treeview",
    executables = [Executable("exoTreeView.py")],
)
```

- Exécuter dans une fenêtre commande
`python setup.py build`
- L'exécutable et dll sont dans le répertoire `build`

- Des modules Python permettent de s'interfacer aux bases de données
MySQL, Postgre, NoSQL, SQLite
- <http://apprendre-python.com/page-database-data-base-donnees-query-sql-mysql-postgre-sqlite>
- En langage PHP l'extension PDO permet d'écrire un seul code d'accès pour des bases de données différentes
En Python l'équivalent est l'ORM (*Object-relational mapping*)
SQLAlchemy