3 PYTHON

1 COMMENTAIRES ET DOCUMENTATION

• **Écrire un commentaire : #** commentaire

• **Écrire de la documentation (docstring) :** (extension autoDocstring sur VS Code)

"""_summary_ : description

Args:

  paramètre1: description du paramètre1

  paramètre2: description du paramètre2

Returns:

  type: valeur de retour

Raises:

  TypeError: condition de l'erreur

"""

Remarque : il faut éviter les accents, même dans les commentaires.

2 TYPES DE VARIABLE

• **Entier :** int, Integer

• **Flottant** (décimal) **:** float

• **Chaine de caractères :** str

• **Booléen** (vrai, faux) **:** bool (True, False)

Remarque : Python gère la déclaration à la volée. Les types sont donc implicites.

3 OPÉRATIONS MATHÉMATIQUES

• **Opérateurs de comparaison :** ==, !=, >, >=, <, <=

• **Opérateurs arithmétiques :** +, -, *, ** (puissance), /, // (pour garder uniquement par partie entière de la division), % (modulo = division entière)

• **Opérateurs logiques :** not, or, and

Remarque : pour utiliser toutes les opérations mathématiques usuelles, importer le module « math »:

from math import sin, pi **OU :** from math import *

4 Opérations sur les variables

• **Déclarer une variable :** variable = valeur

Remarque : on ne peut pas nommer une variable avec un mot réservé. *Liste des 32 mots réservés : and, as, assert, async, await, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield.*

• **Affecter une valeur à une variable :** variable = valeur

• **Affecter différentes variables simultanément :** variable1, variable2 = valeur1, valeur2

• **Permuter les valeurs de deux variables :** variable1, variable2 = variable2, variable1

• **Incrémenter une valeur :** variable = variable + 1 **OU :** variable+=1 (pareil pour les autres opérateurs arithmétiques)

• **Décrémenter une valeur :** variable = variable - 1

5 Transtypage (cast)

• **Convertir une variable en chaine de caractères :** str(variable)

• **Convertir une variable en entier :** int(variable)

• **Convertir une variable en décimal :** float(variable)

• **Convertir une variable en booléen :** bool(variable)

• **Convertir une chaine de caractère/un tuple en liste :** list(chaine/tuple)

• **Convertir une liste en tuple :** tuple(liste)

6 Entrées et sorties

• **Récupérer la saisie de l'utilisateur :** variable = input("texte_à_afficher_à_l'utilisateur")

• **Afficher une variable :** print(variable)

7 Opérations sur les chaines de caractères

• **Obtenir la longueur d'une chaine :** len('chaine')

• **Concaténer deux chaines :** chaine3 = chaine1 + chaine2

• **Répéter une chaine :** chaine2 = chaine1 * nombre

• **Scinder une chaine en liste de mots :** 'chaine'.split('séparateur')

• **Concaténer une liste en une chaine :** 'séparateur'.join(liste)

• **Donner la position d'une sous-chaine dans une chaine :** 'chaine'.find('sous-chaine') (le premier indice vaut 0 ; -1 sera retourné si la sous-chaine n'est pas trouvé)

• **Donner le nombre de sous-chaines dans une chaine :** 'chaine'.count('sous-chaine')

• **Convertir une chaine en minuscules :** 'chaine'.lower()

• **Convertir une chaine en majuscules :** 'chaine'.upper()

• **Convertir la première lettre d'une chaine en majuscule :** 'chaine'.capitalize()

• **Convertir la première lettre de tous les mots en majuscule :** 'chaine'.title()

• **Intervertir les casses d'une chaine :** 'chaine'.swapcase()

• **Supprimer les blancs en début et en fin de chaine :** 'chaine'.strip()

• **Remplacer une sous-chaine1 par une sous-chaine2 :** 'chaine'.replace('sous-chaine1', 'sous-chaine2')

• **Obtenir un caractère d'une chaine :** chaine[position] (commence à 0)

• **Obtenir les caractères d'une chaine de la position1 à la position2 :** chaine[position1:position2-1] (il faut bien enlever 1 à la deuxième position)

• **Obtenir les caractères d'une chaine jusqu'à une position :** chaine[:position-1] (idem)

• **Obtenir les caractères d'une chaine à partir d'une position :** chaine[position:] (idem)

• **Obtenir le dernier caractère d'une chaine :** chaine[-1]

• **Insérer une variable dans une chaine :** chaine = f"{variable}" ou chaine = "{formatage}".format(variable). Ex : {:3d} → 3 entiers, {:02.0f} → 2 chiffres après la virgule, la variable est un nombre flottant

8 STRUCTURES DE DONNÉES

	Liste	**Tuple**	**Set**	**Dictionnaire**
Mutable (différents types)	X		X	X
Ordonné	X	X		X (Python 3.7 et +)
Doublons	X			
Rapide	Lent	Plus rapide que les listes	Plus lent pour l'insertion	• Rapide • Plus lent en insertion et suppression
Modifiable	X			X
Accessibilité des éléments	Index	Index	Index	Clé

8.1 Listes

- **Déclarer une liste vide :** liste = []
- **Déclarer une liste :** liste = [élément1, élément2]
- **Créer une liste de listes :** liste3 = [liste1, liste2]
- **Créer une liste jusqu'à un nombre :** range(nombre+1)
- **Créer une liste d'un nombre à un autre nombre :** range(nombre1, nombre2+1)
- **Ajouter un élément à la fin d'une liste :** liste.append(élément)
- **Trier une liste :** liste.sort()
- **Trier une liste mais à l'inverse :** liste.reverse()
- **Trier une liste par l'élément numéro 1 de sa sous-liste imbriquée :** liste.sort(key=lambda i: i[1])
- **Rechercher l'index d'un élément dans une liste :** liste.index(élément)
- **Supprimer un élément d'une liste :** liste.remove(élément)
- **Supprimer le contenu d'une liste** : liste.clear()
- **Obtenir un élément d'une liste :** liste[position] (commence à 0)
- **Obtenir les éléments d'une liste de la position1 à la position2 :** liste[position1:position2-1] (il faut bien enlever 1 à la deuxième position)
- **Obtenir les éléments d'une liste jusqu'à une position :** liste[:position-1] (idem)
- **Obtenir le dernier élément d'une liste :** liste[-1]
- **Vérifier si un élément appartient à une liste :** élément in liste
- **Répéter un(des) élément(s) dans une liste :** liste = [élément1, élément2] * nombre_de_fois
- **Obtenir la longueur d'une liste :** len(liste)
- **Donner le nombre fois où un élément se trouve dans la liste :** liste.count(élément)
- **Ajouter les éléments de la liste2 à la fin de la liste :** liste.extend(liste2)
- **Manipuler deux chaines comme un dictionnaire :** zip(liste1, liste2)

8.2 Tuples

- **Déclarer un tuple :** tuple = (élément1, élément2)

Remarque : les fonctions s'appliquant aux listes s'appliquent aux tuples (sauf celles qui modifient les données)

8.3 Sets

- **Déclarer un set :** set = {élément1, élément2}
- **Ajouter les éléments à un set :** set.add(élément)

8.4 Dictionnaires

- **Déclarer un dictionnaire vide :** dico = { }
- **Déclarer un dictionnaire :** dico = { "clé": "valeur", "clé": "valeur" }
- **Créer un couple clé-valeur (ou modifier la valeur de l'élément) :** dico["clé"] = "valeur"
- **Supprimer un élément par une clé de dictionnaire :** del dico["clé"]
- **Supprimer un élément et renvoyer une valeur :** dico.pop("clé")
- **Supprimer un élément et renvoyer une clé et une valeur :** dico.popitem("clé")
- **Supprimer tous les éléments du dictionnaire :** dico.clear()
- **Afficher toutes les clés d'un dictionnaire :** dico.keys()
- **Afficher toutes les valeurs d'un dictionnaire :** dico.values()
- **Afficher tous les couples d'un dictionnaire :** dico.items()
- **Déterminer si la clé existe dans le dictionnaire :** dico.has_key() (si la clé est dans le dictionnaire, la fonction renvoie true, sinon false)

8.5 Listes en compréhension

- **Liste en compréhension :** expression qui permet de générer une liste de manière très compacte, équivalente à une boucle for qui construirait la même liste en utilisant la méthode append().

- Syntaxe :

```
result = [valeur for x in séquence condition2]

# est équivalent à :

result = []

for x in séquence:

  condition2:

    result.append(valeur)
```

Remarque : la condition2 peut aller de « if x > 23 » à « for y in séquence2 »

9 STRUCTURES DE CONTRÔLE

9.1 Structures conditionnelles

• **Exécuter un bloc d'instructions si une condition est remplie avec IF :**

```
if condition1:
  instructions
elif condition2:
  instructions
else:
  instructions
```

• **Syntaxe compacte du IF :**

```
variable = valeur1 if condition else valeur2
# est équivalent à :
if condition:
  variable = valeur1
else:
  variable = valeur2
```

• **Exécuter un bloc d'instructions si une condition est remplie avec MATCH** (à partir de Python 3.10) **:**

```
case 0:
  instructions
…
case N:
  instructions
case default:
  instructions
```

• **Vérifier si deux conditions sont vraies en même temps :** if condition1 and condition2:

• **Vérifier si l'une des deux conditions est vraie :** if condition1 or condition2:

• **Vérifier si une variable est égale à 0 / faux / vide :** if not variable:

• **Vérifier si une valeur est présente dans une liste :** if valeur in liste:

• **Vérifier si deux variables pointent vers le même objet :** if variable1 is variable2:

• **Vérifier si deux variables ne pointent pas vers le même objet :** if variable1 is not variable2:

9.2 Boucles

• **Exécuter un bloc d'instructions tant qu'une condition est remplie :** while condition:

9.3 Structures itératives

• **Parcourir une liste/tuple/set :** for valeur in liste/tuple/set:

• **Exécuter un bloc d'instructions sur un nombre défini d'itérations :** for k in range(nombre+1):

• **Parcourir des chiffres de nombre1 à nombre2 :** for k in range(nombre1,nombre2+1):

• **Parcourir les clés d'un dictionnaire :** for k in dico.keys():

• **Parcourir les valeurs d'un dictionnaire :** for k in dico.values():

• **Parcourir les couples (clé/valeur) d'un dictionnaire :** for k in dico.items():

• **Parcourir deux listes à la fois, tel un dictionnaire :** for valeur1, valeur2 in zip(liste1, liste2):

• **Parcourir deux listes à la fois, tel un dictionnaire mais en itérant :** for k, (valeur1, valeur2) in enumerate(zip(liste1, liste2)):

10 FONCTIONS

• **Écrire une fonction :**

```
def fonction(paramètre1, paramètre2 : type = valeur_par_défaut) -> type_retourné :
  """ Documentation de la fonction. """
  instructions
```

Remarques :

— un paramètre peut être une fonction.

— il est préférable de définir les paramètres par défaut en dernier.

• **Appeler une fonction :**

```
fonction(paramètre1=valeur1, valeur2)
```

• **Passer un nombre variable de paramètres dans une fonction et les parcourir avec un(e) tuple/liste :**

```
def fonction(*args):
  for k in args:
```

• **Appeler une fonction avec un nombre variable de paramètres avec un(e) tuple/liste :** fonction(*liste)

• **Passer un nombre variable de paramètres dans une fonction et les parcourir avec un dictionnaire :**

def fonction(**kwargs):

  for k in kwargs.values():

• **Appeler une fonction avec un nombre variable de paramètres avec un dictionnaire :** fonction(**dico)

• **Sortir des valeurs d'une fonction :** return valeur1, valeur2 (un retour de fonction arrête l'exécution de la fonction)

• **Récupérer les valeurs d'une fonction dans des variables :** variable1, variable2 = fonction()

• **Typer les paramètres d'une fonction et le retour :**

def fonction(paramètre1 : type, paramètre2 : type) -> type:

Remarque : c'est une bonne pratique de typer la sortie mais Python n'en tient pas compte.

• **Créer une fonction anonyme, qui n'a pas de nom et qu'on ne va jamais réutiliser** (expression lambda) **:** (lambda paramètre1, paramètre2 = valeur_par_défaut : paramètre1 + paramètre2) (valeur_paramètre1)

• **Appliquer une fonction sur chaque élément d'une liste avec une expression lambda :** list(map(expression_lamda, liste)

• **Filtrer une liste en fonction d'une condition avec une expression lambda :** list(filter(expression_lamda, liste)

• **Réduire une liste en une seule valeur avec une expression lambda :** reduce(expression_lamda, liste)

11 MODULES

• **Importer un module complet** (mauvaise pratique) **:** import module as nom_module_résumé

• **Importer des fonctions particulières d'un module :** from module import fonction1, fonction2

• **Empêcher une partie du code d'être exécutée lorsque le module est importé** (vérification que le fichier est bien exécuté en tant que premier fichier) **:**

if __name__ == "__main__"

• **Voir la liste des fonctions d'un module** (après avoir importé le module) **:** print(dir(module))

• **Utiliser une fonction d'un module** (si importé « import module ») **:** module.fonction()

• **Utiliser une fonction d'un module** (si importé « from module import fonction ») : fonction()

12 PIP : GESTIONNAIRE DE PAQUETS UTILISÉ EN LIGNE DE COMMANDES

• **Rechercher un paquet :** aller sur https://pypi.org/

• **Installer la dernière version de paquets** (qui n'est pas dans la bibliothèque standard) (avant de pouvoir l'importer et l'utiliser comme module) **:** pip install paquet1, paquet2

Remarque : pour installer des paquets uniquement sur un projet particulier, il faut être connecté sur l'environnement virtuel avant de faire « pip install ».

• **Installer une version précise de paquet :** pip install -Iv paquet==version

• **Mettre à jour un paquet vers la dernière version :** pip install -U paquet

• **Mettre à jour un paquet vers une version précise :** pip install -U paquet==version

• **Désinstaller un paquet :** pip uninstall paquet

• **Lister les paquets installés :** pip freeze

• **Sauvegarder la liste des paquets installés dans un fichier :** pip freeze > fichier.txt

• **Installer tous les paquets depuis un fichier :** pip install -r fichier/base.txt

• **Voir les informations d'un paquet installé :** pip show paquet

13 CLASSES ET OBJETS

13.1 Dans le fichier principal

• **Utiliser un fichier de classe dans le fichier principal :** from Classe import *

• **Instancier une classe** (créer un objet à partir de la classe) **:** objet = Classe()

• **Appeler une méthode depuis l'objet :** objet.méthode()

• **Utiliser un attribut depuis l'objet :** objet.attribut

• **Convertir un objet en chaine de caractère :** print(objet)

• **Afficher les attributs d'un objet :** print(dir(objet)) **OU :** print(objet.__dict__)

• **Afficher un objet :** print(objet)

13.2 Dans la classe

• **Déclarer une classe :**

```
class Classe:
    """ Documentation de la classe. """
    instructions
```

• **Définir une méthode dans une classe :** def méthode(self, paramètre1, paramètreN):

• **Appeler une méthode depuis une autre méthode dans la classe :** self.méthode()

• **Instancier une classe** (constructeur) **:** def __init__(self, paramètre1, paramètreN): (la bonne pratique est de déclarer les attributs ici uniquement)

• **Définir un attribut dans une méthode :** self.attribut = valeur

• **Appeler un attribut depuis une autre méthode :** self.attribut (ne fonctionne pas si la variable n'est pas déclaré dans __init__)

• **Convertir un objet en chaine de caractère** (méthode) **:**

```
def __str__(self):
    return "chaine de caractère"
```

• **Afficher les attributs d'un objet** (méthode) **:**

```
def __dir__(self):
  return ["attribut1", "attributN"]
```

OU :

```
def __dict__(self):
  return {"attribut1": "valeur1", "attributN": "valeurN"}
```

• **Afficher un objet** (méthode) **:**

```
def __repr__(self):
  return "Classe(attribut1='valeur1', attribut2='valeur2')"
```

• **Définir un attribut de classe** (attribut statique) **:**

```
class Classe:
  attribut = valeur (avant le __init__)
```

• **Définir une méthode de classe** (méthode statique) **:**

```
@staticmethod
def méthode():
```

• **Appeler une méthode depuis la classe :** Classe.méthode()

• **Utiliser un attribut depuis la classe :** Classe.attribut

13.3 Héritage

• **Créer une classe à partir d'une classe existante** : class ClasseFille(ClasseMère):

• **Appeler une méthode de classe parente dans la classe fille :** super().méthode()

• **Redéfinir une méthode :** définir une méthode avec le même nom dans la classe fille

• **Surcharger une méthode :** appeler la méthode de la classe parente dans la classe fille et ajouter du code

13.4 Visibilité

• **Public :** accessible depuis n'importe où. Ex : self.attribut = valeur

• **Protected :** accessible depuis la classe et ses classes filles. Ex : self._attribut = valeur (c'est juste une convention)

• **Private :** accessible uniquement depuis la classe. Ex : self.__attribut = valeur

13.5 Encapsulation

• **Définir un getter :**

```
def get_attribut(self)
  return self.__attribut
```

OU :

```
@property
def attribut(self)
  return self.__attribut
```

• **Définir un setter :**

```
def set_attribut(self, valeur)
  self.__attribut = valeur
```

OU :

```
@attribut.setter
def attribut(self, valeur)
  self.__attribut = valeur
```

13.6 Polymorphisme

• **Polymorphisme de substitution :** les objets de classes différentes peuvent être utilisés de manière interchangeable, à condition que la classe définie soit la classe parente. Ex :

```
class Animal:
  def make_sound(self):
    print("Animal making sound")

class Dog(Animal):
  def make_sound(self):
    print("Woof")

class Cat(Animal):
  def make_sound(self):
    print("Meow")

def make_animal_sound(animal : Animal):
  animal.make_sound()
```

• **Polymorphisme de surcharge :** on peut redéfinir plusieurs méthodes provenant d'une classe parente pour changer leur comportement

```
class Shape:
  def area(self):
    print("Shape's area")

class Rectangle(Shape):
  def area(self):
    print(f"Rectangle's area: {self.width * self.height}")

class Circle(Shape):
  def area(self):
    print(f"Circle's area: {3.14 * self.radius * self.radius}")
```

• **Polymorphisme d'interface :** on peut définir des méthodes avec le même nom dans des classes différentes qui accompliront des tâches différentes via l'utilisation de classes abstraites

```
class Shape(ABC):
  @abstractmethod
  def area(self):
    pass

class Rectangle(Shape):
  def area(self):
    print(f"Rectangle's area: {self.width * self.height}")

class Circle(Shape):
  def area(self):
    print(f"Circle's area: {3.14 * self.radius * self.radius}")
```

13.7 Abstraction (déprécié)

Remarque : utiliser « from abc import ABC, abstractmethod »

• **Déclarer une classe abstraite** (une classe abstraite ne peut pas être instanciée) **:**

```
class Classe(ABC):
```

• **Déclarer une méthode abstraite** (une méthode abstraite est une méthode qui n'a pas de corps et qui doit être redéfinie dans les classes filles) **:**

```
@abstractmethod
def méthode(self):
```

13.8 Relation entre les objets

• **Composition :** relation entre deux objets où l'un est contenu dans l'autre (si l'objet contenant est détruit, l'objet contenu est détruit). Exemple :

```
class House:
  def __init__(self):
    self.porte = Door()
```

• **Agrégation :** relation entre deux objets où l'un est importé dans l'autre. Exemple :

```
class Voiture:
  …

class Garage:
  def __init__(self, voiture):
    self.voiture = voiture

voiture = Voiture():
garage = Garage(voiture)
```

14 GESTION D'ERREURS

• **Tester un code qui peut poser problème** (try) **et définir les actions à prendre si une exception est rencontrée** (except)**, exécuter du code s'il n'y a aucune erreur** (else) **et exécuter du code dans tous les cas** (finally) **:**

```
try:
  instructions
except Exception: # toutes les exceptions (pas recommandé)
  instructions
except NomException1 as nom1:
  instructions
except NomException2 as nom2:
  instructions
else:
  instructions
finally:
  instructions
```

• **Lever volontairement une exception :** raise NomException, "message_d'erreur"

• **Voir la liste des erreurs possibles à gérer :**

```
import builtins
print(dir(builtins))
```

15 IGNORER UNE PARTIE DE CODE

• **Sortir d'une boucle :** break

• **Sauter la partie d'une boucle où une condition est déclenchée, passer à l'itération suivante** (sans sortir de la boucle) **:** continue

16 RÉFÉRENCES

• Si un objet modifiable est affecté, tout changement sur un objet modifiera l'autre. Ex :

```
liste1 = ['a', 'b']        # liste1 = ['a', 'b']
liste2 = liste1            # liste2 = ['a', 'b']
liste1[1] = 'c'            # liste1 = ['a', 'c'], liste2 = ['a', 'c']
```

• **Faire une vraie copie d'un objet :**

```
import copy
copie = copy.deepcopy(objet_à_copier)
```

17 VARIABLES LOCALES ET GLOBALES

• **Obtenir les variables locales :** locals()

• **Obtenir les variables globales :** globals()

• **Indiquer qu'on utilise une variable globale :** global variable

• Remarque : si, dans une fonction, on affecte une nouvelle valeur à une variable locale qui a le même nom qu'une variable globale, on masque la variable globale et on crée une variable locale.

18 DATES

Remarque : utiliser « from datetime import * »

• **Obtenir la date complète du jour :** datetime.datetime.now()

• **Récupérer le jour d'une date :** date.day

• **Récupérer le mois d'une date :** date.month

• **Récupérer l'année d'une date :** date.year

19 CHIFFRES ALÉATOIRES

Remarque : utiliser « import random »

• **Créer un chiffre aléatoire entier :** chiffre = random.randint(minimum, maximum)

• **Créer un chiffre aléatoire flottant :** chiffre = random.uniform(-minimum, maximum)

20 EXPRESSIONS RÉGULIÈRES

Remarque : utiliser « import re »

• **Créer un pattern à partir d'une expression régulière :** pattern = re.compile(r"expression_régulière")

• **Créer un pattern avec des alternatives :** pattern = re.compile(r"pa(alternative1|alternative2)ttern")

• **Rechercher la première occurrence d'un pattern et sa position dans une chaine :** pattern.search("chaine")

• **Rechercher toutes les occurrences du pattern dans une chaine :** pattern.findall("chaine")

• **Remplacer toutes les occurrences d'un pattern dans une chaine :**

pattern.sub("sous-chaine_qui_remplace", "chaine")

21 FICHIERS

• **Ouvrir un fichier en mode lecture / écriture / ajout :** fichier = open("fichier.txt", "r/w/a")

• **Ouvrir un fichier sans avoir besoin de le fermer en mode lecture / écriture / ajout :** with open("fichier.txt", "r/w/a") as fichier:

• **Écrire une donnée simple dans un fichier :** fichier.write(valeur)

• **Écrire des données multiples dans un fichier :** fichier.writelines(liste)

• **Ajouter des données à la fin d'un fichier :** print >> fichier, données

• **Lire la totalité d'un fichier :** fichier.read(), fichier.readlines()

• **Lire au plus x octets :** fichier.read(x)

• **Lire la ligne suivante :** fichier.readline()

• **Parcourir les lignes d'un fichier :** for ligne in fichier:

• **Trouver si une ligne contient une valeur :** if valeur in ligne: **OU :** if ligne.find(valeur) != -1: **OU:** if ligne.__contains__(valeur)

• **Fermer un fichier :** fichier.close()

22 VENV : MODULE UTILISÉ POUR CRÉER ET GÉRER DES ENVIRONNEMENTS VIRTUELS

• **Installer ven :** py -m pip install --user virtualenv

• **Créer un environnement virtuel :** py -m venv .venv

• **Activer l'environnement virtuel** (sous Powershell) **:** .\venv\Script\Activate.ps1 (ou utiliser le plugin « Python Environment Manager »)

• **Autoriser l'exécution du script « Activate.ps1 »** (sous Powershell en mode administrateur) **:** Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser

23 MODULE « PANDAS » : UTILISÉ POUR GÉRER DES DONNÉES

Remarque : utiliser « pip install pandas » et « import pandas »

• **Afficher les informations générales du tableau de données :** dataframe.info()

• **Récupérer un dataframe à partir d'un fichier CSV :** dataframe = pandas.read_csv(fichier)

• **Obtenir les lignes k à i du dataframe :** dataframe[k:i]

• **Obtenir les x premières lignes d'un dataframe :** dataframe.head(x)

• **Obtenir les x dernières lignes d'un dataframe :** dataframe.tail(x)

• **Tester si un champ commence par une chaine de caractères :** dataframe['champ'].str.startswith('chaine')

• **Obtenir uniquement les colonnes 1 et 2 d'un dataframe :** dataframe[['colonne1', 'colonne2']]

• **Sauvegarder le dataframe dans un CSV :** dataframe.to_csv('fichier_csv', index=false)

• **Trier par ordre croissant une colonne :** dataframe.sort_values(by=['colonne'])

• **Obtenir la donnée la plus fréquente d'une colonne :** dataframe['colonne'].mode().iloc[0]

• **Obtenir uniquement les lignes avec une certaine condition :** dataframe[condition] (exemples : dataframe[dataframe['colonne'] == valeur])

• **Changer le format d'une date :** date = pandas.to_datetime(date, format="%m-%d-%Y")

• **Appliquer une fonction sur chaque ligne d'une colonne :** dataframe = dataframe['colonne'].apply(fonction)

• **Obtenir la valeur moyenne / maximale / minimale (colonne1) d'une colonne (colonne2) :** dataframe.groupby('colonne2')['colonne1'].mean() / .max() / .min()

• **Trouver les doublons d'une colonne :** dataframe[dataframe['colonne'].duplicated(keep=False)].sort_values(by='colonne')

• **Obtenir la répartition de la colonne1 par la colonne2** (exemple : professions par pays) **:**

dataframe.groupby(['colonne2', 'colonne1']).size().unstack()

• **Créer des bornes délimitant des intervalles et les nommer grâce à des étiquettes :**

bornes = [intervalle1, …, intervalleN] (exemple : [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

étiquettes = ['étiquette1', …, 'étiquetteN']

• **Créer une nouvelle colonne qui contiendra des étiquettes en fonction des intervalles et calculé selon une colonne :** dataframe['nouvelle_colonne'] = pandas.cut(dataframe['colonne'], bornes, étiquettes, include_lowest=True)

• **Obtenir le nombre d'occurrences de chaque valeur unique d'une colonne :** dataframe['colonne'].value_counts()

• **Obtenir le pourcentage d'une colonne :** dataframe['colonne'].value_counts(normalize=True) * 100

• **Afficher un histogramme :**

— **importer le module nécessaire :** import matplotlib.pyplot as plt

— **créer une liste pour définir les bornes d'intervalles afin de diviser les données d'une colonne en groupes d'une certaine valeur :** liste = range(0, int(dataframe['colonne'].max()), 10)

— **tracer l'histogramme grâce aux données d'une colonne et à des bornes d'intervalles :** plt.hist(dataframe['colonne'], bins=bornes_d'intervalles, edgecolor='black')

— **ajouter un titre :** plt.title('titre')

— **ajouter une étiquette d'axe x au graphique :** plt.xlabel('étiquette')

— **ajouter une étiquette d'axe y au graphique :** plt.ylabel('étiquette')

— **afficher le graphique :** plt.show()

• Fonctions REGEX de Pandas : https://kanoki.org/2019/11/12/how-to-use-regex-in-pandas/

24 MODULE « PYMYSQL » :

Remarque : utiliser « pip install pymysql » et « import pymysql »

• **Créer une connexion à la base de données :**

connexion = pymysql.connect(hôte, utilisateur, mot_de_passe, base_de_données)

• **Créer un curseur qui va parcourir les éléments de la table :** curseur = connexion.cursor()

• **Parcourir chaque ligne d'un tableau de données :** for ligne in dataframe.iterrows():

• **Insérer des données dans une table de la base de données** (à utiliser dans une boucle) **:**

sql = "INSERT INTO table (champ1, …, champN) VALUES (%s, …, %s)"

curseur.execute(sql, (ligne['colonne1'], …, ligne['colonneN']))

• **Valider les modifications apportées à la base de données :** connexion.commit()

• **Fermer la connexion à la base de données et libérer les ressources associées** (à ne pas oublier) : connexion.close()

25 MODULE « SYS »

Remarque : utiliser « import sys »

• **Vérifier le système Python utilisé :** print(sys.argv)

• **Vérifier la version de Python utilisée :** print(sys.version)

• **Vérifier les chemins d'accès des modules Python :** print(sys.path)

• **Ajouter un chemin d'accès dans le sys :** sys.path.append("chemin")

• **Vérifier les entrées/sorties/erreurs :**

```
for i in (sys.stdin, sys.stdout, sys.stderr):
  print(i)
```

• **Vérifier les modules intégrés au système :** print(sys.builtin_module_names)

• **Faire des appels aux scripts shell :** os.system("bash -c \"read -n 1\"")

26 MODULE « OS »

Remarque : utiliser « import os »

• **Vérifier le système d'exploitation utilisé :** print(os.name) # nt pour Windows

• **Afficher le répertoire de travail :** print(os.getcwd())

• **Changer de répertoire courant :** os.chdir("chemin")

27 FICHIERS À CRÉER LORSQUE L'ON DÉBUTE UN PROJET

• **« LICENCE » :** la licence MIT est bien pour la plupart des projets

• **« .gitignore » :** permet d'éviter de sauvegarder des répertoires inutiles pour le repo git (comme l'environnement virtuel) ou des fichiers qui peuvent être générés automatiquement

• **« README.md » :** explique à quoi sert le projet, sous le format Markdown (on peut le structurer de la sorte : titre du projet, description, prérequis, installation)

28 AUTRE

• **Obtenir le sinus de Pi :** sin(pi)

• **Raccourci pour que les espaces soient au bon endroit dans VS Code :** alt + shift + f

29 UTILISER L'AIDE DE LA DOCUMENTATION

- Lister toutes les classes et les méthodes dans les classes d'une librairie : dir(librairie)
- Afficher la docstring d'une méthode : help(méthode)