Introduction à Python

M1 IEF - 2024

1. Qu'est ce que Python ?

2. La nécessité de l'indentation

3. Qu'est ce que Jupyter ?

4. Quels sont les IDEs (Integrated Development Environments) à ma disposition ?
    1. (Default) IDLE
    2. (Mon choix) Visual Studio Code (VSCode)
    3. PyCharm
    4. Sublime Text
    5. Jupyter
    6. Spyder
    7. Et bien d'autres ...

Plan

01 Les variables et les types

02 Les structures de contrôle

03 Les fonctions

04 Manipuler les dataframes

05 Les séries temporelles

06 Les APIs

1 - Les variables et les types (1/2)

1. Les types
   - Chaines de caractères: **str**
   - Numériques: **int, float, complex**
   - Booleens: **bool**
   - Vide: **None**

2. Les listes : [1, 2.0, 'hi']

3. Afficher un résultat: **print**
   Concaterner du texte avec les fstring : f"{var}"

4. Les modules
   Deux incontournables : pandas (pour les dataframes) et numpy (pour les aspects matriciels)

5. Importer un fichier
   - Chemin relatif vs absolue (./, ../)
   - csv, xlsx
   - Vérifier les types des colonnes

```
# Affecte la valeur 5 à la variable 'a'
a = "5"
a <- "5"

# Affiche le type de 'a’
type(a)

# Incrémente la variable 'a' de 2
a = a + 2
a+=2

# Change le type de 'a'
a = int(a)
a = float(a)

# Affichage dynamique basé sur la valeur de 'a'
print( "La valeur de a est", a)
print( f"La valeur de a est {a} ")

# Importer un module
import pandas as pd
import numpy as np
```

1. **Les dictionnaires**: structure ou des valeurs sont associées à des clés. Un dictionnaire est toujours déclaré avec des **{}**

   myDict = {"key1": values, " key3":values, ...}

   Liste des clés d'un dictionnaire: myDict.keys()
   Liste des valeurs d'un dictionnaire: myDict.values()

2. **Les listes**: structure de données qui contient une collection d'objets Python. Elles peuvent contenir des valeurs de types différents (par exemple entier et chaîne de caractères). Une liste est toujours declarée avec des **[]**

   myList = ["value1", "value2", "value3", ...]

   - Les listes en compréhension: raccourci pour créer une liste par ajout successifs d'éléments

   myListCompr = [val for val in myList]

```python
# Déclarer un dictionnaire
myDict = {"voiture": 4, "vélo": 2, "tricycle": 3}
myDict.keys()
myDict.values()

for key, value in myDict.items():
    print(f"l'élément de clé {key} vaut {value}")

# Déclarer une liste
animaux = ["girafe", "tigre", "singe", "souris"]
animaux[0]

# Exemple de liste en compréhension
[val for val in myDict.values()]

[f"l'élément de clé {key} vaut {value}" for
key,value in myDict.items()]
```

1. Les boucles
   - for : parcours tous les éléments d'un vecteur et execute autant de fois un code donné

     for i in vecteur:
         code

   - while : Exécute un codé donné tant qu'une condition n'est pas remplie

     while condition:
         code

2. Les conditions: execute ou non un code en fonction d'une condition

   if condition:
       code
   elif condition:
       code
   else:
       code

```
# parcours le vecteur c(1,2,3,4) et affiche les valeurs
for i in range(1,5):
  print(i)

# parcours un autre vecteur et affiche les valeurs
for i in ["Bonjour","tout","le","monde","!"]:
  print(i)


# Incrémente i de 1 jusqu'a ce que i vaille 5
i = 1
while i < 5:
  print(i)
  i += 1


# Affichage des valeurs de a sous certaines conditions
a = 7

if a == 7:
  print("a = 7")
elif  a == 10:
  print("a = 10")
else:
  print("a n'est ni égal à 7 ni à 10.")
```

1. Vérifier si un nombre est premier ou pas. Pour savoir si un nombre est divisible par un autre, utilisez le modulo (% en Python)

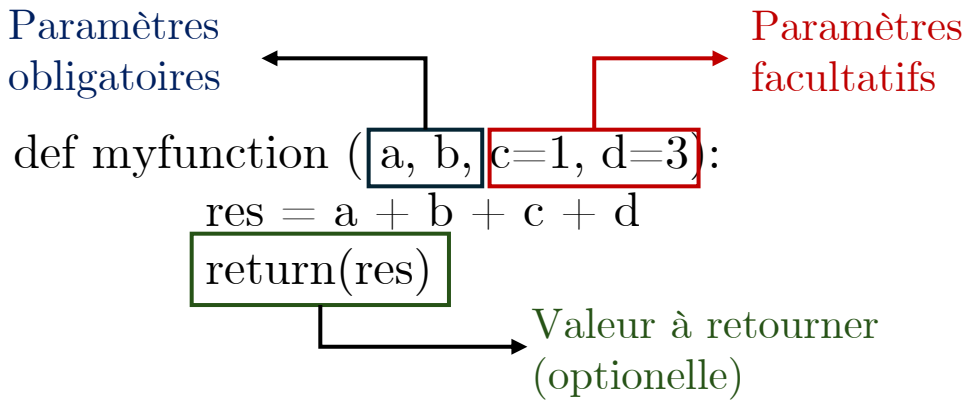
2. Les deux premiers termes de la suite de Fibonacci sont tous deux un. Les termes suivants de la séquence sont trouvés en additionnant les deux termes immédiatement précédents. Écrivez un code qui écrit les n termes de la séquence de Fibonacci pour tout $n \geq 3$

3. Écrivez un code qui simule le lancer d'une pièce avec une probabilité p associée aux piles, jusqu'à obtenir fâce, et renvoie le nombre de piles obtenues. Ensuite, reproduisez l'exécution de ce code 100 fois et afficher le nombre maximum de piles obtenus lors d'une même séquence.

   Pour générer une probabilité entre 0 et 1, utiliser :

   import random
   random.random()

Une fonction informatique est un ensemble d'instructions regroupées sous un nom et s'exécutant à la demande (l'appel de la fonction).




```python
# Déclaration d'une fonction
def myFunction(a:int, b:int, c:list) -> int:
    """ Additionne 3 valeurs
      Args:
        - a: la premiere valeur
        - b: la seconde valeur
        - c: la derniere valeur
    """
    res = a + b + c
    return res

# Différents appels à ma fonction
myFunction(a = 1, b = 2)
myFunction(1, 2)
xx = myFunction(a = 1, c = 2, b = 3)
```

Toutes les variables contenues dans une fonction (paramètres inclus) sont 'locales' et ne sont accessibles que dans cette fonction. Elles sont créées à l'appel de la fonction et detruites à la fin de son exécution. Par opposition aux variables déclarées partout ailleurs dans le code, dites 'globales' et accessibles partout (y compris dans une fonction).

1. Vérifier si un nombre est premier ou pas. Pour savoir si un nombre est divisible par un autre, utilisez le modulo (%% en R)

   => Transformer votre réponse en utilisant une fonction avec un seul parametre **obligatoire** définissant le nombre à vérifier.

2. Les deux premiers termes de la suite de Fibonacci sont tous deux un. Les termes suivants de la séquence sont trouvés en additionnant les deux termes immédiatement précédents. Écrivez un code qui écrit les n termes de la séquence de Fibonacci pour tout $n \geq 3$

   => Transformer votre réponse en utilisant une fonction avec un seul parametre **obligatoire** n définissant le nombre de termes à ajouter à la suite de Fibonacci.

3. Écrivez un code qui simule le lancer d'une pièce avec une probabilité p associée aux piles, jusqu'à obtenir fâce, et renvoie le nombre de piles obtenues. Ensuite, reproduisez l'exécution de ce code 100 fois et afficher le nombre maximum de piles obtenus lors d'une même séquence.

   Pour générer une probabilité entre 0 et 1, utiliser :

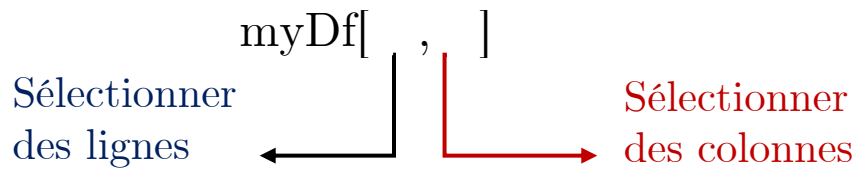
   import random
   random.random()

   => Transformer votre réponse en utilisant une fonction avec deux paramètres **facultatifs**. Un premier pour définir la probabilité associée aux piles (par défaut 0.5), et un second pour le nombre de lancers (par défaut 100).

En python, les dataframes ne sont pas natifs et ils faut les importer via la module « pandas »: import pandas as pd

myDf = pd.DataFrame()

Tout comme en R, c'est un format matriciel amélioré avec des colonnes et des lignes nommées. Un DataFrame peut être créé à partir d'un dictionnaire, d'une liste de listes, ou encore d'un fichier CSV



Pour sélectionner une ou plusieurs lignes, pandas propose plusieurs méthodes :

.loc[] : pour sélectionner par le nom de l'index.

.iloc[] : pour sélectionner par l'indice numérique.

```python
# Création d'un dataframe
data = {
    "Nom": ["Alice", "Bob", "Charlie"],
    "Âge": [24, 27, 22],
    "Ville": ["Paris", "Lyon", "Marseille"]
}
df = pd.DataFrame(data)

# Operations de base
len(df)
df.shape
df.describe().T
df.columns
df.index

df.iloc[1:3,] # les lignes 2 et 3
df[~df.index.isin([1, 2])] # tout sauf les lignes 2 et 3
df.iloc[-1,] # selectionne la dernière ligne
df.iloc[:,1:3] # les colonnes 2 et 3
df.iloc[:2, :2] #Deux premuères lignes et colonnes
df[["Nom", "Âge"]]
df.loc[0, ["Nom", "Âge"]]
df.Nom

# Sous-selection conditionelle
df[df.Age>=25]
```

- Afficher les dimensions d'un dataframe: df.shape
- Concatener deux dataframes côtes à côtes: pd.concat([df1, df2])
- Concatener deux dataframes l'un sous l'autre: pd.concat([df1, df2], axis=1)
- Afficher les noms de colonnes: df.columns
- Afficher les noms de lignes: df.index
- Renommer les colonnes: df.rename(columns={"old":"new"})
- Afficher les N premières lignes: df.head(N)
- Afficher les N dernières lignes: df.tail(N)

```
# Création d'un dataframe
import pandas as pd
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", None)

df.rename(columns={"Nom":"name", "Ville":"city"},
inplace=True)
df.replace({"name":{"Alice":"toto","Bob":"titi"}},
inplace=True)
```

Exercice:

1. Importer les fichiers « eia_data.csv » et « brent.csv »
2. Vérifier qu'ils ont les bonnes dimensions pour les fusionner
3. Concatener les deux dataframes (ne prendre qu'une seule col
4. Renommer les colonnes en :
   c("date","cocpopec","copsopec","paprnonopec","papropec","paprrus","paprus","pascoecd","patcchn","patcnonoecd","patcoecd","patcwld","brent")

- Afficher les valeurs distinctes d'une colonne: df.value_counts()
- Trier un df par rapport à une colonnes: df.sort_values("colname")

- Faire une operation matricielle en ligne ou en colonne (ici la moyenne):
  - En ligne : df.apply(lambdas x: function(x) myfonc(x))
  - En colonne: df.apply(lambdas x: function(x) myfonc(x), axis = 1)

- Faire une operation glissante sur N périodes (ici moyenne mobile):
  - df.rolling(N, min_periods=Y).mean()

- Passer d'un format long à wide et inversement:
  - Long => wide: df.pivot(index=..., columns=...)
  - Wide => long: df.melt(id_vars=...,)

```python
# Importer un fichier csv
df = pd.read_csv("input/eia_data.csv")

# Faire la moyenne des lignes
from statistics import mean
df.iloc[:,1:].apply(lambda x: mean(x))

# Calculer une moyenne mobile 10 jours
df.iloc[:,1:].rolling(10, min_periods=10).mean()

# Passage à un format long
df_melt = df.melt(id_vars="date", var_name="symbol")

# Repasser à un format wide
df_melt.pivot(index="date",
columns="symbol",values="value").reset_index()
```

Une série temporelle est un format matriciel ayant un index au **format date**. Il est ensuite possible d'appliquer un grand nombre d'opération sur ces séries, comme les lags.

Pour ce faire, il faut lui donner un index au format date

```
df.date = pd.to_datetime(df.date)
df.set_index("date"  , inplace=True)
```

Créer un lag: df.shift(N)

Gérer les données manquantes:

- suppression: df.dropna()
- forward fill: df.fillna(method="ffill")
- backward fill: df.fillna(method="bfill")
- remplacer par 0: df.fillna(0)
- interpollation linéaire: df.interpolate()

```python
# Importer un fichier csv
df = pd.read_csv("input/eia_data.csv")

# Le transformer en série temporelle
df.date = pd.to_datetime(df.date, format="%d/%m/%Y")
df.set_index("date", inplace=True)

# Changer la fréquence en trimestrielle (aggregation Moyenne)
df.resample("QS").mean()

# Faire un lag de 1
df.shift(1)
```

1. Importer les fichiers brent.csv et eia_data.csv depuis le repertoire input et renommer les colonnes
2. Transformer ces deux structures en série temporelle avec un format xts
3. Fusionner les données dans une variable « base »
4. Formater « base » en trimestrielle
5. Créer une variable « base_ts » qui est la même que « base » mais au format ts
6. Effectuer un lag de 5 trimestres sur la colonne « brent » de « base » et l'ajouter dans la base
7. Calculer une moyenne mobile 6 trimestres sur cette même colonne et l'ajouter dans la base
8. Calculer la croissance annuelle du brent et l'ajouter dans la base
9. Créer un indicateur qui vaut 1 si la croissance du brent est positive ou nulle, 0 sinon, et l'ajouter dans la base
10. Sauvegarder la base dans un fichier « baseFinale.csv » situé dans un repertoir « output »

Les API (interface de programmation d'application) sont des mécanismes qui permettent à deux composants logiciels de communiquer entre eux à l'aide d'un ensemble de définitions et de protocoles

1. API REST (Representational State Transfer)
   Ce sont les API les plus demandées et les plus flexibles que l'on retrouve aujourd'hui sur le web. Le client adresse des demandes au serveur sous forme de données. Le serveur utilise les données du client pour exécuter des fonctions internes et renvoie les données de sortie au client. REST définit un ensemble de fonctions comme GET, PUT, DELETE, etc., que les clients peuvent utiliser pour accéder aux données du serveur.

   Les clés d'API permettent de vérifier le programme ou l'application à l'origine de l'appel d'API. Elles identifient l'application et s'assurent qu'elle dispose des droits d'accès requis pour effectuer l'appel d'API particulier.

Fonctionnement d'une API REST

1. Construction de l'URL

Base url : Adresse racine de l'API
Chemin : Chemin d'accès au service souhaité
Options : Informations additionnelles souhaitées lors de l'exécution de l'appel. La déclaration de la première option commence par un **?** et les options suivantes sont ajoutées avec un **&**
Header : Non utilisable par navigateur. Ce sont des informations à envoyer à l'API complémentaires ou substituables aux options.

2. Méthodes d'appel (get, post, put, ...) : Méthode à utiliser lors de l'appel API. Un même chemin répondra différemment si on l'appel avec des méthodes différentes. Il faut consulter la documentation de l'API pour savoir les méthodes associées aux chemins définis.

3. Response status : Une fois l'exécution faite, l'API renverra un code permettant de déterminer si l'opération c'est bien passée ou s'il y a eu une erreur (voir liste des status)

Le **json** (JavaScript Object Notation) est un format standard **de données textuelles** utilisé pour représenter des données structurées de façon semblable aux objets Javascript. Il est habituellement utilisé pour structurer et transmettre des données sur des sites web. Il concurrence XML.

Il comprends :

1. deux types composés :
   - des objets JavaScript, ou ensembles de paires « nom » (ou « clé ») / « valeur ».
   - des listes ordonnées de valeurs (tableau).

2. quatre types scalaires :
   - des booléens : prend la valeur true ou false.
   - des nombres : un nombre décimal signé qui peut contenir une part fractionnable ou élevée à la puissance. Le JSON n'admet pas les nombres inexistants (NaN), et ne distingue pas les entiers et les flottants.
   - des chaînes de caractères : une séquence de 0 ou plus caractères Unicode. À l'instar des clés, elles sont obligatoirement entourées de guillemets.
   - La valeur null : qualifie l'absence de valeur.


```
{
  object: "time_series",
  id: "FRED/DTWEXEMEGS/REG_EM",
  dataset: "fred",
  dataset_name: "Federal Reserve Economic Data",
  dataset_description: "FRED is an online database consisting of
  provider_name: "Federal Reserve Bank (FED)",
  free: true,
  terms_of_use: "https://fred.stlouisfed.org/legal/#fred-terms",
  symbol: "DTWEXEMEGS",
  name: "Nominal Emerging Market Economies U.S. Dollar Index",
  description: null,
  frequency: "D",
  unit: "Index Jan 2006=100",
  adjustment: "NSA",
  category: null,
  country_id: "REG_EM",
  country_name: "Emerging Markets",
  country_short_name: "Emerging Markets",
- data: [
  - {
      timestamp: "2006-01-02",
      value: 100.9386
    },
  - {
      timestamp: "2006-01-03",
      value: 100.8318
    },
  - {
      timestamp: "2006-01-04",
      value: 100.4582
    },
  - {
      timestamp: "2006-01-05",
      value: 100.3368
    },
  - {
      timestamp: "2006-01-06",
      value: 100.1888
    },
```

Pour faire un appel API, on utiliser request.get (ou .put, ou .post, ...)

Il est également possible d'attacher un header :

resp = requests.get(url, headers=...)

Pour avoir le code de status renvoyé par l'appel :

resp.status_code

Dans le cas d'une méthode poste, il est souvent demandé des données lors de l'appel (sous la forme d'un dictionnaire):

resp = requests.get(url, headers=..., json=...)

Si le code de retour est dans les 200, alors il faut décoder le json renvoyé (dans le cas ou le chemin retourne bien du json):

resp.json()

```python
import requests

def get(path, apikey):
    headers = {'Content-Type': 'application/json',
'Accept':'application/json'}
    headers.update({"Authorization": "Bearer {}".format(apikey)})

    try:
        res = requests.get(f"https://api.taceconomics.io/v2
/{path}", headers=headers)
        if res.status_code == 200:
          res = res.json()
          return res

    except Exception as e:
        print(e)
        return None

    return None

res = get("mydata", "myapikey")
df = pd.DataFrame(res["data"])
df.set_index('timestamp', inplace=True)
df.columns = [code.lower()]
```