PYTHON

1. **Ubuntu :**
   * Ubuntu est un système d'exploitation basé sur Linux. Il est une distribution Linux populaire, offrant un système d'exploitation complet.
   * Il est utilisé pour exécuter des applications, des serveurs, et pour le développement logiciel.
   * Ubuntu peut être installé sur un ordinateur comme système d'exploitation principal.
2. **WSL (Windows Subsystem for Linux) :**
   * WSL est un sous-système de Windows qui permet d'exécuter un environnement Linux sur un système Windows.
   * Il permet d'utiliser des outils et des applications Linux directement sur Windows, ce qui est utile pour les développeurs qui ont besoin de travailler avec des outils Linux tout en utilisant Windows comme système d'exploitation principal.
3. **Windows PowerShell :**
   * Windows PowerShell est un interpréteur de commandes et un environnement de script intégré à Windows.
   * Il est utilisé pour l'automatisation des tâches, la gestion système, et l'exécution de scripts PowerShell.
   * PowerShell est plus orienté vers les tâches d'administration système.
4. **Python :**
   * Python est un langage de programmation polyvalent utilisé pour le développement de logiciels, le scripting, la science des données, l'automatisation, etc.
   * Il peut être exécuté sur différentes plates-formes, y compris Windows, Linux, et macOS.
   * Pour exécuter des scripts Python, vous avez besoin d'un interpréteur Python installé sur votre système.
5. **Visual Studio Code (VS Code) :**
   * VS Code est un éditeur de code source léger, open source et extensible développé par Microsoft.
   * Il est utilisé par les développeurs pour écrire du code dans divers langages de programmation, y compris Python.
   * VS Code dispose d'une large gamme d'extensions qui permettent de personnaliser et d'étendre ses fonctionnalités pour différents besoins de développement.

En résumé, Ubuntu est un système d'exploitation Linux complet, WSL permet d'exécuter un environnement Linux sur Windows, Windows PowerShell est un interpréteur de commandes Windows, Python est un langage de programmation, et Visual Studio Code est un éditeur de code source. Vous pouvez les utiliser ensemble pour développer des applications Python sur Windows en utilisant des outils Linux via WSL, mais chacun a son propre rôle et utilité distinc

CLASS

class Class\_name:

    def \_\_init\_\_(self, object) :

        self.object = attribute

    def method\_name (self, object2):

        self.object\_from\_class.action(object2)

the '.update' methods that allows us to add multiple key-value pairs from a dictionnary to another one

Numpy’s functions

1. **Creating Arrays**:
   * **np.array()**: Create an array from a Python list or tuple.
   * **np.zeros()**: Generate an array filled with zeros.
   * **np.ones()**: Generate an array filled with ones.
   * **np.arange()**: Create an array with evenly spaced values.
2. **Array Manipulation**:
   * **np.reshape()**: Change the shape of an array.
   * **np.concatenate()**: Concatenate arrays along a specific axis.
   * **np.split()**: Split an array into multiple sub-arrays.
   * **np.transpose()**: Transpose an array.
3. **Mathematical Operations**:
   * **np.sum()**, **np.mean()**, **np.min()**, **np.max()**: Perform operations on arrays (sum, mean, min, max, etc.).
   * **np.dot()**: Perform matrix multiplication between arrays.
   * **np.linalg.inv()**, **np.linalg.det()**: Linear algebra operations like finding inverses and determinants.
4. **Random Number Generation**:
   * **np.random.rand()**: Generate random numbers from a uniform distribution.
   * **np.random.randn()**: Generate random numbers from a normal distribution.
   * **np.random.randint()**: Generate random integers within a specified range.
5. **Statistical Functions**:
   * **np.histogram()**: Generate a histogram from an array.
   * **np.correlate()**: Compute the correlation between two arrays.

All the numpy random number generation :

\*\*Simple Random Data\*\*:

- `numpy.random.rand()`: Generate random numbers from a uniform distribution over `[0, 1)`.

- `numpy.random.randn()`: Generate random numbers from a standard normal distribution.

- `numpy.random.randint()`: Generate random integers within a specified range.

- `numpy.random.random\_sample()`: Generate random floats in the half-open interval `[0.0, 1.0)`.

- `numpy.random.random()`: Same as `random\_sample()`.

- `numpy.random.choice()`: Generate a random sample from a given 1-D array.

- `numpy.random.bytes()`: Generate random bytes.

2. \*\*Permutations\*\*:

- `numpy.random.shuffle()`: Modify a sequence in-place by shuffling its contents.

3. \*\*Distributions\*\*:

- `numpy.random.normal()`: Draw random samples from a normal (Gaussian) distribution.

- `numpy.random.uniform()`: Draw samples from a uniform distribution.

- `numpy.random.binomial()`: Draw samples from a binomial distribution.

- `numpy.random.poisson()`: Draw samples from a Poisson distribution.

- `numpy.random.gamma()`: Draw samples from a gamma distribution.

- `numpy.random.exponential()`: Draw samples from an exponential distribution.

- `numpy.random.chisquare()`: Draw samples from a chi-square distribution.

4. \*\*Random Generator\*\*:

- `numpy.random.default\_rng()`: Create a `Generator` instance for the new random number generator.

5. \*\*Random Sampling\*\*:

- `numpy.random.shuffle()`: Modify a sequence in-place by shuffling its contents.

- `numpy.random.permutation()`: Randomly permute a sequence or return a permuted range.

These functions offer different ways to generate random numbers or samples from various distributions. You can use these functions based on your requirements for randomness, distribution, and sequence manipulation.

CALCUL Stock Price avec Daily Return

l = int(input("Price after how many days ? ")) #'int()' to set it as integers inputs

initial\_price = float(input("enter initial price of your stock : ")) #'float()' to set it as floats

stock\_price= initial\_price

for i in range(2,l+1) :  #the range function in Python excludes the end value ( i€[2;l[ ), so with l+1 i€[2;l]

    stock\_price = stock\_price \* (1+daily\_return[i-1])

**Mais pourquoi i-1 ??**

En Python et dans de nombreux langages de programmation, les indices des éléments d'une liste ou d'un tableau commencent généralement à zéro. Ainsi, le premier élément d'une liste a un indice de 0, le deuxième élément a un indice de 1, et ainsi de suite.

Dans le contexte de votre code, la liste `daily\_return` contient les rendements journaliers générés par `np.random.normal()`. Lorsque vous accédez aux éléments de cette liste à l'intérieur d'une boucle, la variable de boucle `i` commence à partir de 1 (selon votre code `range(1, l + 1)`), mais les indices dans la liste commencent à partir de 0.

Par exemple, si vous souhaitez accéder au premier élément de la liste `daily\_return`, vous utiliseriez `daily\_return[0]`, pas `daily\_return[1]`.

Ainsi, lors de l'itération à travers la liste, l'indice correct pour accéder à l'élément correspondant dans la liste serait `i - 1`. Cela garantit que vous commencez à l'indice 0 et que vous atteignez le dernier indice (`l - 1` ou `n - 1` dans ce cas) pour parcourir tous les éléments de la liste.

Par conséquent, dans la ligne `stock\_price = stock\_price \* (1 + daily\_return[i - 1])`, `i - 1` est utilisé pour accéder correctement au rendement quotidien correspondant au jour dans la liste `daily\_return` à mesure que la boucle progresse à partir de 1.