Python

Master1 **IMSD**

IFRISSE

Pierre Claver OUEDRAOGO

Tables de matières

Aperçu du cours

Objectifs du cours

Dans ce cours, nous allons étudier le langage Python , pourquoi Python, ses utilisations et ses avantages . De façon pratique, nous allons voir les différentes fonctionnalités de Python, comment l'utiliser pour exploiter son potentiel.

Ce cours aborde l'ensemble des fonctionnalités de base et utiles de Python. L'objectif étant de compléter vos connaissances en programmation informatique.

A la fin de ce cours, vous devrez être capable d'écrire , de comprendre du code Python et d'utiliser les librairies Python pour faire de l'analyse de données.

Prérequis

Pour suivre ce cours dans de bonnes conditions, il est essentiel que vous possédiez des bases de la programmation informatique. Ce qui est le cas puisque vous avez déjà eu des cours de programmation.

Unité 0 : Généralités

A la base conçu pour pouvoir automatiser certaines taches répétitives et contraignantes afin de les laisser exécuter par une machine plutôt qu'à la main, Python a été crée dans les années1990 par Guido Van Rossum (qui a aussi crée le navigateur Grail).

Avantages

Le langage python a d'énormes avantages car :

- Python est portable;
- Python est gratuit;
- Python est un langage interprété;
- Python est orienté objet.

La syntaxe de Python est simple, très puissant et utilisé dans le monde scientifique.

Python a également une grosse base de communauté .

Unité 0 : Généralités

Domaines d'application

Ce point est assez délicate, car il existe de nombreuses applications pour Python.

Python permet de faire des choses vraiment incroyables, mais si l'on peut catégoriser ces utilisations en 3 grandes catégories, ce serait :

- Développement web
- Data Science – incluant le machine Learning, l'analyse de données ainsi que la visualisation de données
- Scripting

Développement web

Python à travers ses Framework Django et Flask (les plus populaires), nous permets de construire des applications web. un Framework web facilite la création d'une architecture commune côté serveur.

Unité 0 : Généralités

Développement web

Cela comprend le mappage de différentes URL sur des fragments de code Python, le traitement de la base de données et la génération de fichiers HTML vus par les utilisateurs dans leurs navigateurs.

Data Science et Data Visualization

Il existe de nombreux frameworks et de nombreuses bibliothèques de **machine learning** en Python.

Scikit-learn et TensorFlow sont parmi les plus populaires :

- **scikit-learn** est livré avec certains des algorithmes les plus populaires d'apprentissage automatique intégrés.
- **TensorFlow** est plutôt une bibliothèque de bas niveau qui vous permet de créer des algorithmes de machine learning personnalisés.

Unité 0 : Généralités

Data Science et Data Visualization

Python nous permet de faire de l'analyse et de la Data visualization . A travers sa bibliothèque MATPLOTLIB , on peut aisément faire de la Data visualisation.

C'est une des bibliothèque les plus populaire de python :

- Elle est facile à prendre en main;
- De nombreuses bibliothèques sont basées sur cette dernière, comme **Seaborn**. Donc, apprendre Matplotlib vous aidera à apprendre ces autres bibliothèques par la suite.

Unité 0 : Généralités

Le Scripting et les applications Bureau

L'écriture de scripts fait généralement référence à l'écriture de petits programmes conçus pour automatiser des tâches simples. Un script, c'est un mot balise pour dire petit bout de programme écrit un peu comme ça au fil de l'eau pour automatiser des tâches rébarbatives.

Parmi des exemples de scripts courants, on peut citer :

- Le tri automatique de mails;
- La manipulation de fichiers (renommage, nettoyage, …);
- La modification d'images (trims, filtres, …).

Python permet de faire des applications bureau . Vous pouvez en créer en utilisant **Tkinter** ou **PyQt**, mais cela ne semble pas non plus être le choix le plus populaire car d'autres langage de programmation existent et sont plus populaires pour ce type d'application.

Unité 0 : Généralités

Les Versions de Python

Les versions de Python développées depuis sa création et jusqu'en 2009 sont des versions 2.x.

Depuis 2009, Python est passé à la version 3.X

La dernière version en date de **24 mars 2022** est la **version 3.10.4**

Unité 0 : Généralités

En résumé

Python est un langage de programmation assez généraliste, On peut ~~**tout**~~ faire avec python : des sites et applications web, des applications mobiles, des scripts personnels, des applications de bureau, de l'analyse de données, implémenter des algorithme de Machine learning et même des jeux vidéo !

Par exemple :

- Instagram est codé en Python ;
- c'est l'un des langages principaux utilisés chez Google ;
- Netflix utilise Python pour l'élaboration de ses algorithmes de recommandation ;
- l'application bureau de Dropbox est développée en Python ,etc..

Concernant la data, Python est le langage le plus utilisé, en particulier pour les traitements et manipulations de données, la data visualisation et le développement d'algorithmes d'intelligence artificielle.

Unité 1 : Mise en place de l'environnement de travail Python

Python est un langage ***interprété***.

Pour coder en Python, un seul programme va être strictement obligatoire : ***l'interpréteur Python*** qui va pouvoir être utilisé de manière dynamique et nous afficher les résultats de nos codes immédiatement.

L'interpréteur Python est disponible en téléchargement gratuit sur le site officiel du langage, dans la rubrique téléchargement : ***https://www.python.org/downloads/***

Une fois l'interpréteur téléchargé et installé, vous aurez besoin d'un **éditeur de texte** pour écrire vos lignes de code.

On en dénombre plusieurs dont **pyCharm** spécialement conçu pour écrire du code Python.

Vous pouvez le télécharger via le lien suivant : **https://www.jetbrains.com/pycharm/**

Une autre façon d'écrire du code Python est d'utiliser **Jupyter.**

Unité 1 : Mise en place de l'environnement de travail Python

Jupyter est une application web utilisée pour programmer dans plus de 40 langages de programmation, dont Python, Julia, Ruby, R, ou encore Scala.

Jupyter permet de réaliser des calepins ou notebooks, c'est-à-dire des programmes contenant à la fois du texte et du code en Python, R...

Ces calepins sont utilisés en science des données pour explorer et analyser des données.

Installez Python via Anaconda

Anaconda est une distribution scientifique de Python : c'est-à-dire qu'en installant Anaconda, vous installerez Python, Jupyter Notebook et des dizaines de packages scientifiques, dont certains indispensables à l'analyse de données !

Vous pouvez télécharger la distribution Anaconda correspondant à votre système d'exploitation, en Python version
https://www.anaconda.com/distribution/

3 : Nous allons utiliser Jupyter Notebook pour la suite de ce cours.

Unité 1 : Mise en place de l'environnement de travail Python

Installation Python via Anaconda

Une fois téléchargé, installé en utilisant simplement les options par défaut. Ces options sont suffisantes pour ce cours

Jupyter Notebook

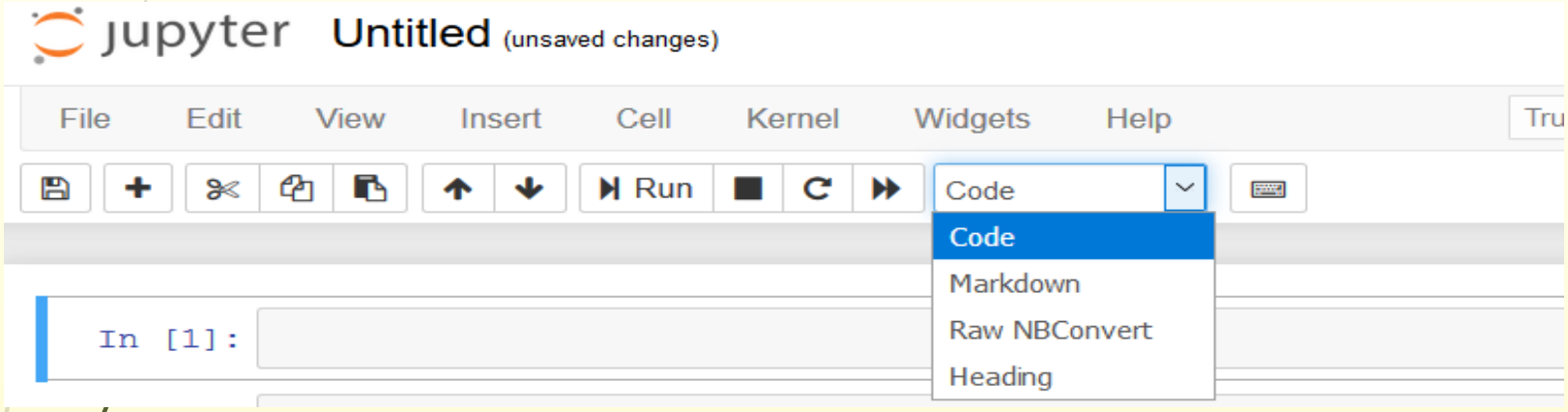
Jupyter Notebook est un outil puissant qui permet aux utilisateurs du langage Python de créer et de partager des documents interactifs contenant du code dynamique et exécutable, des visualisations de contenus, des textes de documentation et des équations. Le terme "notebook" est lié au caractère intrinsèque de l'outil qui permet d'écrire des petits bouts de code exécutable (appelés "cellules"), de les documenter pour expliquer ce qu'ils font et d'afficher les données résultant de leur exécution. Tout cela est stocké dans un document partageable avec d'autres utilisateurs.

Faites vos premiers pas avec Jupyter

Il existe 4 types de cellules différentes avec Jupyter : **code**, **markdown**, **row nbconvert** et **heading**.

Unité 1 : Mise en place de l'environnement de travail Python

Lorsque vous cliquez sur une cellule, vous pouvez changer le type simplement en le sélectionnant dans la liste, comme dans l'exemple ci-dessous :



Voyons en détails chaque type :

- **Code** – la cellule de code classique. Celle-ci est réservée pour l'écriture et l'exécution de code Python ! Vous pouvez exécuter votre code en cliquant sur le bouton **Run**.

Unité 1 : Mise en place de l'environnement de travail Python

Voyons en détails chaque type :

- **Heading** – ce type est légèrement obsolète. Jusqu'alors, il servait pour définir des titres, mais les **Markdown** couvrent aujourd'hui cette fonctionnalité. Il est amené à disparaître dans les prochaines versions de Jupyter.

- **Raw nbconvert** – permet de contrôler le formatage du document lors d'une conversion du notebook dans un autre format. Vous n'allez pas être amené à l'utiliser dans le cadre de ce cours.

- **Markdown** – cellule de texte qui sert essentiellement à la documentation du notebook, pour y rédiger des commentaires, des titres, des équations, etc. Ce type permet de structurer votre texte en utilisant les balises HTML ou la syntaxe Markdown.

Que ce soit pour du ***Markdown*** ou du **code**, il suffit d'écrire votre texte/code et de l'exécuter via le bouton **Run** pour voir le résultat.

Unité 1 : Mise en place de l'environnement de travail Python

On pourrait par exemple créer un titre. Pour cela, il suffit d'ajouter un **#** devant votre texte. De façon similaire à ce que l'on peut avoir avec un logiciel de traitement de texte, **un seul # correspondra à un titre de niveau 1, deux # à un titre de niveau 2, etc.**

Vous pouvez aussi décider de mettre votre texte en **gras** ou en **italique**. Pour cela, il suffit tout simplement d'entourer le texte souhaité :

- par ****** pour mettre en **gras** ;
- par une simple ***** pour mettre en ***italique***.

Unité 1 : Mise en place de l'environnement de travail Python

Par exemple, la cellule Markdown suivante :

```
# Un titre de niveau 1
Voici un peu de texte pour présenter un markdown
## titre de niveau 2
**Ceci** est très important
## autre titre de niveau 2
Mais *cela* également !
```

donnera le résultat suivant :

Un titre de niveau 1

Voici un peu de texte pour présenter un markdown

titre de niveau 2

Ceci est très important

autre titre de niveau 2

Mais *cela* également !

Unité 2 : Les variables et types de données Python

Les variables

Une variable, dans le domaine de la programmation informatique, est un conteneur qui sert à stocker une valeur.

Comment déclarer une variable en Python ?

Python ne possède pas de syntaxe particulière pour créer ou "déclarer" une variable : les variables Python sont automatiquement créées au moment **où on leur assigne une valeur**.

Pour créer une variable en Python, on va donc devoir choisir un nom et affecter une valeur à ce nom, c'est-à-dire stocker une valeur dans notre variable.

Le choix du nom pour nos variables est libre en Python.

Unité 2 : Les variables et types de données Python

Comment déclarer une variable en Python ?

Il faut cependant respecter les règles usuelles suivantes :

- Le nom doit commencer par une lettre ou par un underscore ;
- Le nom d'une variable ne doit contenir que des caractères alphanumériques courants (pas d'espace dans le nom d'une variable si de caractères spéciaux comme des caractères accentués ou tout autre signe) ;
- On ne peut pas utiliser certains mots qui possèdent déjà une signification spéciale pour le langage (on parle de mots "réservés").
- Notez que les noms de variables en Python sont sensibles à la casse, ce qui signifie qu'on va faire une différence entre l'emploi de majuscules et de minuscules : un même nom écrit en majuscules ou en minuscules créera deux variables totalement différentes.

Unité 2 : Les variables et types de données Python

Affecter une valeur à une variable en Python

Pour affecter ou "assigner" une valeur à une variable, nous allons utiliser un opérateur qu'on appelle opérateur d'affectation ou d'assignation et qui est représenté par le **signe =.** Attention, le **signe =** ne signifie pas en informatique l'égalité d'un point de vue mathématique : **c'est un opérateur d'affectation.**

Afficher la valeur d'une variable

En python, nous allons pouvoir utiliser la fonction **print()** pour afficher le contenu d'une variable. Il suffit de passer en paramètre de cette fonction le nom de la variable.

Type de données

Python définit de nombreux types de données qu'on va pouvoir stocker dans nos variables et manipuler à loisir ensuite : nombres entiers, décimaux, complexes, chaines de caractères, booléens, listes, tuples, dictionnaires, etc.

Unité 2 : Les variables et types de données Python

Type de données

Python définit trois types de valeurs numériques supportées :

- Le type **int** qui représente tout entier positif ou négatif ;
- Le type **float** qui représente les nombres décimaux et certaines expressions scientifiques comme le e pour désigner une exponentielle par exemple;
- Le type **complex** qui représente les nombres complexes ou nombres imaginaires et qui se sert de la lettre j pour représenter la partie imaginaire d'un nombre.

le type de données "**nombre complexe**" ou **complex** représente les nombres complexes. On va utiliser la lettre **j** ou **J** pour représenter la partie complexe d'un nombre. Comme la plupart d'entre vous n'auront que rarement affaire aux nombres complexes et que ce cours n'est pas un cours de mathématiques.

Unité 2 : Les variables et types de données Python

Le type str ou chaine de caractères

Pour définir une chaine de caractères ou pour stocker une chaine de caractères dans une variable, il faudra l'entourer de guillemets simples ou doubles droits. Si notre chaine de caractères contient elle même des guillemets simples (apostrophes) ou doubles, il faudra les échapper les uns ou les autres en fonction du délimiteur choisi pour qu'ils soient correctement interprétés car sinon Python pensera qu'ils servent à indiquer la fin de la chaine. **Le caractère d'échappement en Python est l'antislash \.**

Le type de valeurs bool ou booléen

Le type de valeur booléen est un type qui ne contient que deux valeurs qui servent à représenter deux états. Les deux valeurs sont **True** (vrai) et **False** (faux). **Attention en Python à bien indiquer des majuscules car dans le cas contraire Python ne reconnaitra pas ces booléens.**

Pour stocker un booléen dans une variable, il ne faut pas utiliser de guillemets : si on les utilise, ce seront les chaines de caractères "True" et "False" qui seront stockés et on ne va pas pouvoir effectuer les mêmes opérations.

Unité 2 : Les variables et types de données Python

Concaténation : l'opérateur **+** est un opérateur de concaténation et pas d'addition.

Pour connaitre le type de valeur stockée dans une variable, on peut utiliser la fonction Python **type().** On va passer la variable à tester en argument de cette fonction (c'est-à-dire écrire le nom de la variable entre les parenthèses de la fonction). La fonction **type()** va alors tester la valeur contenue dans la variable et renvoyer le type de cette valeur.

Unité 2 : Les variables et types de données Python

Les opérateurs

Un opérateur est un signe ou un symbole qui va nous permettre de réaliser une opération. **Le signe =** par exemple est en Python l'opérateur d'affectation simple : il permet d'affecter une valeur à une variable.

Ci-dessous quelques opérateurs :

Opérateurs Arithmétiques

Opérateur	Nom
+	Addition
–	Soustraction
*	Multiplication
/	Division
%	Modulo
**	Puissance
//	Division entière

Unité 2 : Les variables et types de données Python

Les opérateurs de chaines

Les opérateurs de chaines vont nous permettre de manipuler des données de type **str** (chaines de caractères) et par extension des variables stockant des données de ce type.

Python met à notre disposition deux opérateurs de chaine : l'opérateur de **concaténation +** et **l'opérateur de répétition ***.

L'opérateur de concaténation va nous permettre de mettre bout à bout deux chaines de caractères afin d'en former une troisième, nouvelle.

L'opérateur de répétition va nous permettre de répéter une chaine un certain nombre de fois.

Les opérateurs d'affection simple et composés Python

Unité 2 : Les variables et types de données Python

Les opérateurs d'affection simple et composés Python

Python reconnait également des opérateurs d'affectation qu'on appelle "**composés**" et qui vont nous permettre d'effectuer deux opérations à la suite : une première opération de calcul suivie immédiatement d'une opération d'affectation.

Opérateur	Exemple	Equivalent à	Description
=	x = 1	x = 1	Affecte 1 à la variable x
+=	x += 1	x = x + 1	Ajoute 1 à la dernière valeur connue de x et affecte la nouvelle valeur (l'ancienne + 1) à x
-=	x -= 1	x = x – 1	Enlève 1 à la dernière valeur connue de x et affecte la nouvelle valeur à x
*=	x *= 2	x = x * 2	Mutliplie par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
/=	x /= 2	x = x / 2	Divise par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
%=	x %= 2	x = x % 2	Calcule le reste de la division entière de x par 2 et affecte ce reste à x
//=	x //= 2	x = x // 2	Calcule le résultat entier de la division de x par 2 et affecte ce résultat à x
**=	x **= 4	x = x ** 4	Elève x à la puissance 4 et affecte la nouvelle valeur dans x

Unité 3 : Les Tableaux en Python

Les tableaux

Un tableau est une structure de données représentant une séquence finie d'éléments auxquels on peut accéder efficacement par leurs positions ou leurs indices dans la séquence. En python , nous allons pouvoir manipuler les tableaux de plusieurs façon. Les plus utilisés sont les **Liste** et les **dictionnaires.**

Les Listes

Les listes sont des **collections**. En fait, leur nom est plutôt explicite, puisque ce sont des objets capables de contenir d'autres objets de n'importe quel type. On peut avoir une liste contenant plusieurs nombres entiers (1, 2, 50, 2 000 ou plus, peu importe), une liste contenant des flottants, une liste contenant des chaînes de caractères... et une liste mélangeant ces objets de différents types.

Les listes sont des objets ordonnés, c'est-à-dire qu'à chaque élément de la liste est associé un nombre correspondant à son ordre dans la liste. Ce nombre est appelé indice et il **démarre à 0 (et non à 1 !).** Le premier élément est donc associé à l'indice 0, le second à l'indice 1, etc.

Unité 3 : Les Tableaux en Python

Les Listes

Déclarer une liste est assez similaire à la déclaration de n'importe quelle variable vue jusque là : via un nom auquel on associe une liste d'éléments à stocker dans ce nom.

Par exemple, voici la liste contenant les noms de 4 clients :

```python
nomClient = ['Georges Dupont', 'Luc Martin', 'Lucas Anderson', 'Alexandre Petit']
```

Une fois la liste est créée, vous pouvez effectuer deux opérations basiques :

- **accéder** à une valeur à un indice donné ;
- **changer** la valeur à un indice donné.

Dans les deux cas, l'écriture se compose du nom de la **liste** suivi par **[**, la valeur de l'indice et **]**.

Exemple : si vous avez fait une erreur sur le nom du premier client et que vous voulez corriger son nom :

Unité 3 : Les Tableaux en Python

Les Listes

Exemple : si vous avez fait une erreur sur le nom du premier client et que vous voulez corriger son nom :

```python
# assigner la valeur 'Georges Dupond' au premier nom dans notre liste
# c'est donc l'indice 0, car les indices commencent à 0 en python !
nomClient[0] = 'Georges Dupond'
```

Pour l'afficher, vous pouvez écrire la ligne suivante :

```python
print(nomClient[0])
```

Python permet également d'utiliser des **indices négatifs** pour accéder à un élément ou le modifier. **L'indice -1** correspond au **dernier élément** de la liste, **-2** à l'**avant-dernier**, et ainsi de suite. Vous pouvez également accéder à un intervalle d'indices en utilisant **l'opérateur :**, **1:3** permettra par exemple d'accéder aux **éléments 2 à 4**.

Unité 3 : Les Tableaux en Python

Les Listes

```python
# afficher le dernier élément
print(nomClient[-1])

# accéder du second élément au 3ème
print(nomClient[1:3])

# accéder à tous les éléments du début jusqu'au second
print(nomClient[:2])
```

Nous avons ici manipulé des listes de chaînes de caractères, mais vous pouvez faire la même chose avec le montant sur le compte de chaque individu :

```python
montantCompte = [10000, 150, 300, 1800.74]
```

Vous l'aurez probablement noté, mais dans notre dernière liste, les trois premiers éléments sont des entiers, alors que le dernier est un décimal. Comme nous l'avons dit un peu plus haut, vous pouvez très bien **stocker plusieurs objets différents dans une même liste.**

Unité 3 : Les Tableaux en Python

Les Listes

Par exemple, la liste suivante est totalement valide :

```python
listeEtrange = [4, 10.2, "Georges Dupond", ["une autre liste", 1]]

# afficher le 4ème élément de la liste
print(listeEtrange[3])
```

Ajoutez ou supprimez des éléments : les méthodes de listes

les listes sont totalement modifiables, que ce soit le nombre d'éléments, l'ordre de ceux-ci, etc. Grâce aux différentes méthodes de listes, on peut :

- chercher un élément spécifique dans la liste ;
- ajouter un nouvel élément à la fin ;
- insérer un nouvel élément à un indice spécifique ;
- supprimer un élément de la liste.

Unité 3 : Les Tableaux en Python

Les Listes

créer une liste vide en Python et ensuite ajouter les éléments un à un via la méthode **append** :

```python
liste = []
liste.append(7)
liste.append(5)
print(liste) # => [7, 5]
```

1. La première instruction crée une liste vide nommée très originalement liste.
2. Vous ajoutez ensuite l'entier 7 à la fin de la liste. Python va donc l'ajouter à l'indice 0.
3. Finalement, vous ajoutez l'entier 5, qui sera stocké à la suite, à l'indice 1.

Unité 3 : Les Tableaux en Python

Les Listes

Voici d'autres méthodes qu'il est indispensable de connaître autour des listes :

insert : pour insérer un nouvel élément à une position spécifique. Par exemple, **liste.insert(1, 12)** insérera l'entier 12 à l'indice 1, déplaçant l'ancien élément 1 à l'indice 2 et ainsi de suite ;

extend : similaire à append, mais avec une autre liste. D'une certaine façon, cela permet de concaténer plusieurs listes ensemble ;

remove : cherche l'élément donné dans la liste et supprime la première occurrence rencontrée. Par exemple, si vous souhaitez supprimer 5 de votre liste, vous pouvez utiliser :**liste.remove(5);**

index : cette méthode permet de trouver l'indice de la première occurrence d'un élément à chercher dans notre liste ;

mot clé del : pour supprimer un élément selon son indice.

Unité 3 : Les Tableaux en Python

Les Listes

```python
liste = []
liste.append(7) # -> [7]
liste.append(5) # -> [7, 5]
liste.insert(1,12) # [7, 12, 5]
liste[0] = 4 # -> [4, 12, 5]
liste.remove(12) # [4, 5]
liste.index(5) # affiche 1
liste.extend([1, 2, 3]) # [4, 5, 1, 2, 3]
del liste[3] # [4, 5, 1, 3]
```

Décomposons ces quelques lignes :

les trois premières lignes correspondent à ce qui a été vu avant ;

- vous ajoutez ensuite l'entier 12 à l'indice 1. L'ancienne valeur en position 1 est déplacée en position 2 ;
- vous remplacez ensuite la valeur à l'indice 0 par 4 ;
  avec la méthode .remove(),
- vous retirez l'entier 12 de notre liste ;
- vous demandez ensuite l'indice du premier élément 5 dans notre liste (ici en seconde position, donc retourne 1) ;

Unité 3 : Les Tableaux en Python

Les Listes

- vous ajoutez la liste [1, 2, 3] à la suite de notre liste initiale ;
- et vous supprimez finalement l'élément situé en position 4 dans notre liste.

Cela vous laisse finalement avec la **liste finale : [4, 5, 1, 3].**

NB : La **fonction len()** vous permet de récupérer la taille de votre liste :

python

```python
liste = [1, 2, 3]
len(liste) # affichera 3
```

Unité 3 : Les Tableaux en Python

Les dictionnaires

Les dictionnaires sont un autre type d'objet, similaire aux listes.

En effet, un dictionnaire est une **liste** d'éléments organisés via **un système de clés**. Avec un vrai dictionnaire, vous regardez à un nom pour accéder à sa définition. En programmation, ce nom correspond à la **clé** et la définition à la **valeur** qui y est associée. C'est une association **clé-valeur**.

On pourrait ainsi avoir : noms de clients bancaires et le solde du compte en banque associé à chacun.

Georges Dupont	Luc Martin	Lucas Anderson	Alexandre Petit
10000	150	300	1800.74

Chaque clé dans un dictionnaire doit être **unique**. On utilise généralement des chaînes de caractères pour définir les clés, mais ce n'est pas une obligation en soi !

Unité 3 : Les Tableaux en Python

Déclarez un dictionnaire

Les listes et dictionnaires sont déclarés de façon similaire, à la différence qu'un dictionnaire **utilise des accolades** au lieu **des crochets**, et qu'il faut déclarer les **associations clé-valeur :**

python

```python
comptes = {"Georges Dupont": 10000, "Luc Martin": 150, "Lucas Anderson": 300, "Alexandre Petit": 1800.74}
print(comptes["Luc Martin"]) # -> 150
```

La dernière ligne va afficher la valeur associée à la clé "Luc Martin", qui est bien 150.

Manipulez les éléments d'un dictionnaire

Voici les opérations fréquemment réalisées avec des dictionnaires :

- accéder à la valeur d'un élément ;
- ajouter un nouvel élément (valeur-clé) ;
- supprimer un élément via sa clé.

Unité 3 : Les Tableaux en Python

Manipulez les éléments d'un dictionnaire

On peut accéder à une valeur ou la modifier via la même notation qu'avec les listes. Cette notation vous permet même d'ajouter des éléments avec les dictionnaires, contrairement aux listes.

Voyez cela avec l'exemple suivant :

```python
comptes['Georges Dupont'] -= 2000 # je soustrais 2000 au compte de Georges
comptes['Cyril Andreje'] = 1000 # j'ajoute un nouvel individu dans mon dictionnaire
print(comptes['Cyril Andreje']) # j'affiche la valeur du compte de Cyril
```

Finalement, vous pouvez supprimer un élément via la méthode **pop()**, en précisant la clé de l'élément que vous voulez supprimer.

```python
comptes.pop('Luc Martin') # supprime Luc Martin de notre dictionnaire
```

Pour finir, de la même façon qu'avec les listes, vous pouvez utiliser la fonction **len()** pour garder le contrôle sur l'évolution de votre dictionnaire :

```python
len(comptes) # -> 3
```

Unité 4 : Les Structures de contrôles Python

Il existe différents types de structures de contrôle. Les deux types de structures les plus communément utilisées sont les structures de contrôle conditionnelles qui permettent d'exécuter un bloc de code si une certaine condition est vérifiée et les structures de contrôle de boucle qui permettent d'exécuter un bloc de code en boucle tant qu'une condition est vérifiée.

Pour pouvoir vérifier ces conditions, nous allons fréquemment faire appel à des opérateurs de comparaison et des opérateurs logiques.

Voici ci-dessous les différents opérateurs de comparaison disponibles en Python ainsi que leur signification :

Opérateur	Définition
==	Permet de tester l'égalité en valeur et en type
!=	Permet de tester la différence en valeur ou en type
<	Permet de tester si une valeur est strictement inférieure à une autre
>	Permet de tester si une valeur est strictement supérieure à une autre
<=	Permet de tester si une valeur est inférieure ou égale à une autre
>=	Permet de tester si une valeur est supérieure ou égale à une autre

Unité 4 : Les Structures de contrôles Python

Structures de contrôle conditionnelles

Les structures de contrôle conditionnelles (ou plus simplement conditions) vont nous permettre d'exécuter différents blocs de code selon qu'une condition spécifique soit vérifiée ou pas.

Nous allons très souvent utiliser les conditions avec des variables : selon la valeur stockée dans une variable, nous allons vouloir exécuter un bloc de code plutôt qu'un autre.

Python nous fournit les structures conditionnelles suivantes :

- La condition **if** ("si") ;
- La condition **if…else** ("si…sinon") ;
- La condition **if…elif…else** ("si…sinon si… sinon") .

Nous allons étudier et comprendre l'intérêt de chacune de ces conditions dans la suite de cette leçon.

Unité 4 : Les Structures de contrôles Python

La condition if en Python

La structure conditionnelle **if** est une structure de base qu'on retourne dans de nombreux langages de script. Cette condition va nous permettre d'exécuter un code si (et seulement si) une certaine condition est vérifiée.

la syntaxe générale d'une condition if est :

```
if condition :
    code à exécuter
```

Pensez bien à indiquer le **:** et à bien **indenter** le code qui doit être exécuté si la condition est vérifiée sinon votre condition ne fonctionnera pas.

La condition if… else en Python

Avec la condition if, nous restons relativement limités puisque cette condition nous permet seulement d'exécuter un bloc de code si que le résultat d'un test soit évalué à True.
La structure conditionnelle **if…else (« si… sinon » en français**) est plus complète que la condition **if** puisqu'elle nous permet d'exécuter un premier bloc de code si un test **renvoie** True ou un autre bloc de code dans le cas contraire.

Unité 4 : Les Structures de contrôles Python

La condition if… else en Python

La syntaxe d'une condition **if…else** va être la suivante :

```
if condition :
        code à exécuter
else :
        code à exécuter
```

La condition if… elif… else en Python

La condition **if…elif…else (« si…sinon si…sinon »)** est une structure conditionnelle encore plus complète que la condition if…else qui va nous permettre cette fois-ci d'effectuer autant de tests que l'on souhaite et ainsi de prendre en compte le nombre de cas souhaité.

En effet, nous allons pouvoir ajouter **autant** de **elif** que l'on souhaite entre le **if de départ et le else de fin et chaque elif** va pouvoir posséder son propre test ce qui va nous permettre d'apporter des réponses très précises à différentes situations

Unité 4 : Les Structures de contrôles Python

La condition if… elif… else en Python

La syntaxe va être la suivante :

```
if condition :
        code à exécuter
elif condition :
        code à exécuter
elif condition :
        code à exécuter
else :
        code à exécuter
```

Il faut cependant faire attention à un point en particulier lorsqu'on utilise une structure **Python if… elif… else** : le cas où **plusieurs elif** possèdent un test évalué à True par Python. Dans ce cas là, vous devez savoir **que seul le code du premier elif (ou du if si celui-ci est évalué à True) va être exécuté**. En effet, Python sort de la structure conditionnelle dans son ensemble sans même lire ni tester la fin de celle-ci dès qu'un cas de réussite à été rencontré et que son code a été exécuté.

Unité 4 : Les Structures de contrôles Python

Opérateurs d'appartenance ou d'adhésion

Python met à notre disposition **deux opérateurs** d'appartenance qui vont nous permettre de tester si une certaine séquence de caractères ou de valeurs est présente ou pas dans une valeur d'origine. Ces opérateurs ne vont fonctionner qu'avec des séquences (chaines de caractères, listes, etc.) et ne vont donc pas marcher avec des valeurs de type numérique par exemple.

L'opérateur **in** permet de tester si une certaine séquence de caractères ou de valeurs est présente dans une valeur d'origine et renvoie True si c'est le cas.

```
>>> prenoms = ["Pierre", "Mathilde", "Florian", "Thomas"]
>>> if "Pierre" in prenoms:
...     print("Pierre est dans la liste")
...
Pierre est dans la liste
```

L'opérateur **not in** permet au contraire de tester si une certaine séquence de caractères ou de valeurs n'est pas présente dans une valeur d'origine et renvoie True si c'est le cas.

```
>>> ages = [29, 27, 30, 29]
>>> if 31 not in ages:
...     print("Personne n'a 31 ans")
...
Personne n'a 31 ans
```

Unité 4 : Les Structures de contrôles Python

Ordre de priorité des opérateurs

Opérateur	Description
()	Opérateur de groupement
**	Elévation à la puissance
~	Opérateur d'inversion de bit
*, /, %	Opérateurs arithmétiques multiplication, division et modulo
<<, >>	Décalage de bits à gauche ou à droite
&	Définit chaque bit à 1 si les deux bits valent 1
^	Définit chaque bit à 1 si seulement l'un des deux bits vaut 1
|	Définit chaque bit à 1 si au moins l'un des deux bits vaut 1
==, !=, <, <=, >, >=, in, not in, is, is not	Opérateurs de comparaison, d'appartenance et d'identité
not	Opérateur logique (booléen) inverse ou "non"
and	Opérateur logique (booléen) "et"
or	Opérateur logique (booléen) "ou"

Unité 4 : Les Structures de contrôles Python

Les boucles Python

Les boucles vont nous permettre d'exécuter plusieurs fois un bloc de code, c'est-à-dire d'exécuter un code « en boucle » tant qu'une condition donnée est vérifiée.

Lorsqu'on code, on va en effet souvent devoir exécuter plusieurs fois un même code. Utiliser une boucle nous permet de n'écrire le code qu'on doit exécuter plusieurs fois qu'une seule fois.

Nous allons ainsi pouvoir utiliser les boucles pour parcourir les valeurs d'une variable de liste ou pour afficher une suite de nombres.

Nous avons accès à deux boucles en Python :

- La boucle **while** ("tant que…") ;
- La boucle **for** ("pour…").

Unité 4 : Les Structures de contrôles Python

La boucle Python While

La boucle **while** va nous permettre d'exécuter un certain bloc de code **« tant qu'une » condition donnée est vérifiée**. Sa syntaxe est la suivante :

While condition **:**

code à exécuter

Exemple :

```
>>> x = 0
>>> while x < 10:
...     print(x)
...     x += 1
...
0
1
2
3
4
5
6
7
8
9
```

Unité 4 : Les Structures de contrôles Python

La boucle Python for

La boucle Python **for** possède une logique et une syntaxe différente de celles des boucle for généralement rencontrées dans d'autres langages.

En effet, la boucle for Python va nous permettre d'itérer sur les éléments d'une séquence (liste, chaine de caractères, etc.) selon leur ordre dans la séquence.

La condition de sortie dans cette boucle va être implicite **: on sortira de la boucle après avoir parcouru le dernier élément de la séquence**.

La syntaxe de cette boucle va être la suivante :

```
>>> liste = [7, "Pierre", "trail", 29]
>>> for i in liste:
...     print(i)
...
7
Pierre
trail
29
```

Unité 4 : Les Structures de contrôles Python

La fonction range()

On va pouvoir utiliser la fonction **range()** pour **itérer** sur une suite de nombres avec une boucle **for**. Cette fonction permet de générer une suite de valeurs à partir d'une certain nombre et jusqu'à un autre avec un certain pas ou intervalle.
Dans son utilisation la plus simple, nous allons nous **contenter de passer un nombre en argument (entre les parenthèses) de range().** Dans ce cas, la fonction générera une suite de valeurs de **0** jusqu'à ce **nombre – 1** avec un **pas** de **1**. **range(5) par exemple génère les valeurs 0, 1, 2, 3 et 4**.

Si **on précise deux nombres** en arguments de cette fonction, le premier nombre servira de point de départ pour la génération de nombres tandis que le second servira de point d'arrivée (en étant exclus). **range(5, 10)** par exemple permet de **générer les nombres 5, 6, 7, 8 et 9.**

Finalement, on peut préciser **un troisième et dernier nombre en argument** de **range()** qui nous permet de préciser son pas, c'est-à-dire l'écart entre deux nombres générés. Ecrire **range(0, 10, 2)** par exemple permet de générer les nombres **0, 2, 4, 6 et 8.**
On va pouvoir utiliser la fonction range() plutôt qu'une variable de type séquence avec nos boucles for pour itérer sur une suite de nombres.

Unité 4 : Les Structures de contrôles Python

Les instructions break et continue

Les instructions **break** et **continue** sont deux instructions qu'on retrouve dans de nombreux langages et qui sont souvent utilisées avec les boucles mais qui peuvent être utilisées dans d'autres contextes.

L'instruction **break** permet **de stopper l'exécution d'une boucle lorsqu'une certaine condition est vérifiée**. On l'inclura souvent dans une condition de type if.

```
for i in range(10):
    print("debut iteration", i)
    print("bonjour")
    if i == 2:
        break
    print("fin iteration", i)
print("apres la boucle")
```

Résultat :

Affichage après exécution :

```
debut iteration 0
bonjour
fin iteration 0
debut iteration 1
bonjour
fin iteration 1
debut iteration 2
bonjour
apres la boucle
```

Unité 4 : Les Structures de contrôles Python

Les instructions break et continue

L'instruction **continue** permet elle **d'ignorer l'itération actuelle de la boucle et de passer directement à l'itération suivante.** Cette instruction va donc nous permettre d'ignorer toute ou partie de notre boucle dans certaines conditions et donc de personnaliser le comportement de notre boucle.
Exemple :

```
for i in range(4):
    print("debut iteration", i)
    print("bonjour")
    if i < 2:
        continue
    print("fin iteration", i)
print("apres la boucle")
```

Résultat :

```
debut iteration 0
bonjour
debut iteration 1
bonjour
debut iteration 2
bonjour
fin iteration 2
debut iteration 3
bonjour
fin iteration 3
apres la boucle
```

Unité 5 : Les Fonctions en Python

Les Fonctions

Une **fonction** est un bloc de code nommé. Une fonction correspond à un ensemble d'instructions créées pour effectuer une tâche précise, regroupées ensemble et qu'on va pouvoir exécuter autant de fois qu'on le souhaite en "l'appelant" avec son nom. Notez "qu'appeler" une fonction signifie exécuter les instructions qu'elle contient.

Définition d'une fonction - def

La syntaxe Python pour la définition d'une fonction python est la suivante :

```
def nom_fonction(liste de paramètres):
    bloc d'instructions
```

Vous pouvez choisir n'importe quel nom pour la fonction que vous créez, à **l'exception des mots-clés réservés** du langage, et à la condition de n'utiliser **aucun caractère spécial ou accentué** (le caractère souligné **« _ » est permis**). Comme c'est le cas pour les noms de variables, on utilise par convention des minuscules, notamment au début du nom (les noms commençant par une majuscule seront réservés aux classes).

Unité 5 : Les Fonctions en Python

Corps de la fonction

Comme les **instructions if, for et while**, l'instruction **def** est une instruction composée. La ligne contenant cette instruction se termine obligatoirement par un **deux-points :,** qui introduisent un bloc d'instructions qui est précisé grâce à **l'indentation.** Ce bloc d'instructions constitue le **corps de la fonction.**

Fonction sans paramètre

```
def compteur3():
    i = 0
    while i < 3:
        print(i)
        i = i + 1

print("bonjour")
compteur3()
compteur3()
```

En entrant ces quelques lignes, nous avons défini une fonction très simple qui compte jusqu'à 2. Notez bien les parenthèses, les deux-points, et l'indentation du bloc d'instructions qui suit la ligne d'en-tête (c'est ce bloc d'instructions qui constitue le corps de la fonction proprement dite).

Unité 5 : Les Fonctions en Python

Fonction sans paramètre

Après la définition de la fonction, on trouve le programme principal qui débute par l'instruction print("bonjour"). Il y a ensuite au sein du programme principal, l'appel de la fonction grâce à compteur3().

```
def compteur3():
    i = 0
    while i < 3:
        print(i)
        i = i + 1

print("bonjour")
compteur3()
compteur3()
```

Affichage après exécution :

```
bonjour
0
1
2
0
1
2
```

Nous pouvons maintenant réutiliser cette fonction à plusieurs reprises, autant de fois que nous le souhaitons. Nous pouvons également l'incorporer dans la définition d'une autre fonction.

Unité 5 : Les Fonctions en Python

Fonction sans paramètre

```
def compteur(stop):
    i = 0
    while i < stop:
        print(i)
        i = i + 1

compteur(4)
compteur(2)
```

Exécuter

Affichage après exécution :

```
0
1
2
3
0
1
```

Pour tester cette nouvelle fonction, il nous suffit de l'appeler avec un argument.

Unité 5 : Les Fonctions en Python

Fonction sans paramètre

NB: Dans l'exemple ci-dessus, l'argument que nous passons à la fonction compteur() est le contenu de la variable a. A l'intérieur de la fonction, cet argument est affecté au paramètre stop, qui est une tout autre variable.

Notez donc bien dès à présent que :

- Le nom d'une variable que nous passons comme argument n'a rien à voir avec le nom du paramètre correspondant dans la fonction.
- Ces noms peuvent être identiques si vous le voulez, mais vous devez bien comprendre qu'ils ne désignent pas la même chose (en dépit du fait qu'ils puissent contenir une valeur identique).

Unité 5 : Les Fonctions en Python

Fonction avec plusieurs paramètres

La fonction suivante utilise **trois paramètres** : **start** qui contient la valeur de départ, **stop** la borne supérieure exclue comme dans l'exemple précédent et **step** le pas du compteur.

```
def compteur_complet(start, stop, step):
    i = start
    while i < stop:
        print(i)
        i = i + step

compteur_complet(1, 7, 2)
```

Exécuter

Affichage après exécution :

```
1
3
5
```

Unité 5 : Les Fonctions en Python

Fonction avec plusieurs paramètres

NB :

- Pour définir une fonction avec plusieurs paramètres, il suffit d'inclure ceux-ci entre les parenthèses qui suivent le nom de la fonction, en les séparant à l'aide de virgules.

- Lors de l'appel de la fonction, les arguments utilisés doivent être fournis dans le même ordre que celui des paramètres correspondants (en les séparant eux aussi à l'aide de virgules). Le premier argument sera affecté au premier paramètre, le second argument sera affecté au second paramètre, et ainsi de suite.
- On peut également definir une fonction en spécifiant bien une valeur de retour avec le mot clé **return**

Exemple :

```
def cube(w):
    return w**3
```

Unité 6 : Modules et librairies Python pour l'analyse de données

Supposons que vous ayez besoin de calculer la racine carrée d'un nombre dans le cadre d'un de vos notebooks. Il n'existe pas de fonction racine carrée native en Python. Vous pourriez naturellement l'écrire vous-même, mais, il y a sûrement eu un tas de personnes qui se sont déjà posé la même question. Et devinez quoi ? L'un d'eux a déjà écrit la fonction et l'a enregistrée dans un module !

Un module en Python

Un module est un ficher Python contenant un ensemble de fonctions, de classes et de variables prédéfinies et fonctionnelles.
classes :

- carré – défini par la longueur de son côté,
- triangle – défini par la longueur de ses trois côtés,
- cercle – défini par son rayon,etc. ;

variables :

pi : constante indispensable pour calculer l'aire d'un cercle, égale à 3,1415...,
phi : constante représentant le nombre d'or, égale à 1,6180... ;

Unité 6 : Modules et librairies Python pour l'analyse de données

fonctions :

- **aire** : qui prend en paramètre un objet géométrique (carré, triangle, etc.) et calcule son aire.
- **angles** : qui prend en paramètre un triangle, et calcule les angles internes de ce dernier ,etc.

Vous pouvez naturellement définir toutes ces choses dans votre notebook, mais cela ne ferait que l'alourdir. Le mieux est de stocker tout cela dans un fichier Python externe, que vous allez ensuite importer dans votre notebook : c'est un **module** !

Pour pouvoir **importer** un module, vous allez avoir besoin du **mot clé** **import**.

Après avoir fait cela, vous pouvez utiliser les différents éléments définis dans votre module.

Tous les éléments inclus dans le **module geometry** peuvent être utilisés via la notation **nomModule.fonction()** ou encore **nomModule.variable**.

Voici un exemple simplifié d'un **module geometry** :

Unité 6 : Modules et librairies Python pour l'analyse de données

python

```python
'''
Module geometry.py
'''
# variables
pi = 3.14159265359
phi = 1.6180

# fonction qui calcule l'aire
def aire(obj):
    if type(obj) == carre:
        return obj.a**2

# definitions de quelques classes
class carre(object):
    def __init__(self,a):
        self.a = a

class triangle(object):
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c
```

```python
import geometry
```

Après avoir fait cela, vous pouvez utiliser les différents éléments définis dans votre module :

python

```python
car = geometry.carre(4)
tri = geometry.triangle(3, 6, 5)

print(geometry.pi) # -> 3.14159265359

geometry.aire(car) # -> 16
```

Unité 6 : Modules et librairies Python pour l'analyse de données

Les packages

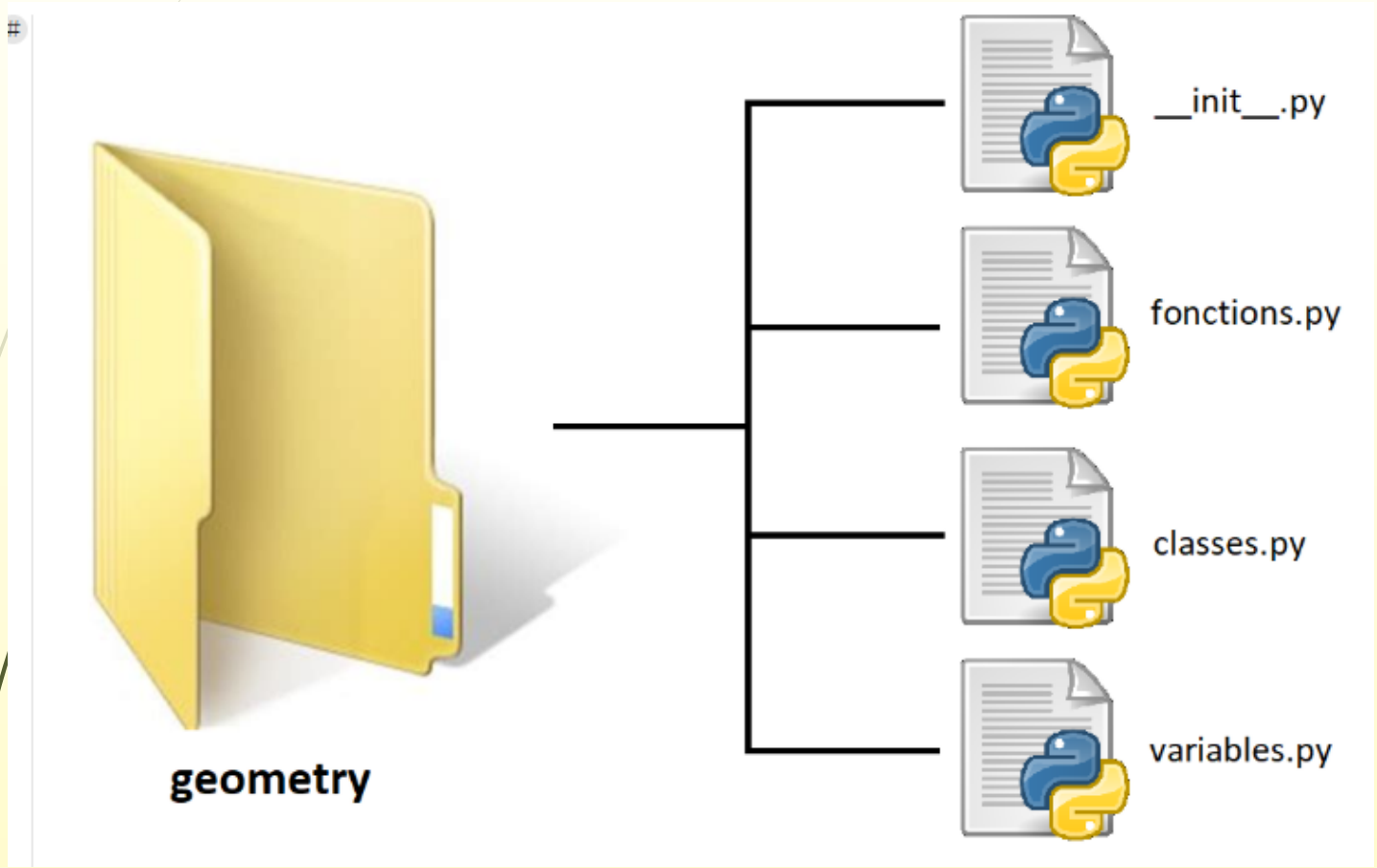
Un **package** (appelé parfois **librairie**) est une collection, **un ensemble de modules Python**. Comme vous l'avez vu ci-dessus, un module est un fichier Python. Un package est simplement un dossier contenant plusieurs fichiers Python (.py) et un fichier additionnel **nommé __init__.py**. Ce dernier différencie un package d'un dossier lambda contenant uniquement des codes Python. Par exemple, vous auriez pu stocker votre module geometry dans trois fichiers différents au lieu d'un seul :

- un pour les classes : classes.py ;
- un pour les variables : variables.py ;
- un pour les fonctions : fonctions.py.

On aurait dans ce cas le dossier suivant :

Unité 6 : Modules et librairies Python pour l'analyse de données

Les packages



Vous aurez besoin d'utiliser l'opérateur . pour accéder au module, après avoir importé le package :

Unité 6 : Modules et librairies Python pour l'analyse de données

Les packages

python

```python
import geometry # import all the geometry package

print(geometry.variables.pi) # -> 3.1415...
car = geometry.classes.carre(4)
geometry.fonctions.aire(car) # -> 16
```

Ou vous pouvez également importer seulement un module du package :

python

```python
import geometry.variables as var # importer uniquement ce qui est défini dans variables.py

print(var.pi) # -> 3.1415...
```

Pour revenir à votre problématique initiale (avoir une fonction **racine carrée**), il y a par exemple le **package numpy** qui en propose une – et bien d'autres choses ! Nous allons l'étudier dans la suite de ce cours.

Unité 6 : Modules et librairies Python pour l'analyse de données

Les packages

Les packages sont omniprésents dans l'analyse de données avec Python. En effet, de nombreux packages ont été créés spécifiquement pour répondre aux problématiques du domaine. Au fur et à mesure de votre parcours, vous allez être amené à :

- manipuler vos données pour en faciliter l'analyse ;
- réaliser différents graphiques pertinents représentant le comportement de vos données ;
- utiliser des méthodes statistiques ;
- faire tourner des algorithmes de **machine learning** plus ou moins compliqués ;etc.

Et pour réaliser tout cela, il vous sera indispensable de maîtriser les différents objets et fonctions issus des packages correspondants.

Unité 6 : Modules et librairies Python pour l'analyse de données

Exemple de module : random

En Python, le module **random** contient plusieurs fonctions pour pouvoir générer des nombres ou des suites de **nombres aléatoires.**

Premièrement, importez votre module random. Le nom du package en Python est... random :

```python
import random
```

La fonction de base de génération de nombre aléatoire s'appelle... **random()** également (quelle originalité ;)). Elle va générer un float aléatoire compris entre 0 et 1 non inclus. Réalisez un exemple simple en affichant 3 nombres aléatoires :

```
In [2]: for i in range(3):
            print(random.random())

0.36276563106916404
0.159050577131982
0.19264162524981843
```

Unité 6 : Modules et librairies Python pour l'analyse de données

Exemple de module : random

Vous avez d'autres fonctions vous permettant de générer un nombre aléatoire dans un intervalle donné :

- **uniform(a, b)** : va générer un float aléatoire compris entre a et b;
- **randint(a, b)** : comme son nom le suggère, celle-ci est similaire à uniform, sauf que le nombre aléatoire généré est cette fois-ci un entier !

Si vous voulez sélectionner un élément aléatoirement dans une liste, une solution un peu naïve pourrait être de tirer l'indice aléatoirement. Le module **random** va un peu plus loin en proposant une fonction permettant de faire la sélection directement sur la liste : **la fonction choice.**

```
In [5]: liste = ['one', 'two', 'three', 'four', 'five']
        for i in range(3):
            print(random.choice(liste))

        one
        four
        three
```

Unité 6 : Modules et librairies Python pour l'analyse de données

Exemple de module : random

L'évolution de celle-ci est la **fonction choices**, permettant cette fois-ci de sélectionner un échantillon de la liste initiale, **avec remise** :

```
In [6]: print(random.choices(liste, k=2))
        print(random.choices(liste, k=3))

        ['five', 'three']
        ['two', 'one', 'two']
```

On parle alors de sous-échantillonnage. La fonction correspondante, pour un **échantillon sans remise**, est **sample** :

```
In [7]: print(random.sample(liste, 2))
        print(random.sample(liste, 3))

        ['two', 'one']
        ['one', 'three', 'two']
```

En analyse de données, cette notion de sous-échantillonnage est primordiale, elle permet de sélectionner un échantillon d'une population initiale. En statistique, un échantillon est un ensemble d'individus représentatifs d'une population. Le recours à un sous-échantillonnage répond en général à une contrainte pratique (manque de temps, de place, coût financier…) ne permettant pas l'étude exhaustive de toute la population.

Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Numpy

La bibliothèque **NumPy** permet d'effectuer des calculs numériques avec Python. Elle introduit une gestion facilitée des tableaux de nombres.

NumPy (diminutif de Numerical Python) fournit une interface pour stocker et effectuer des opérations sur les données.

D'une certaine manière, les tableaux Numpy sont comme les listes en Python, mais Numpy permet de rendre les opérations beaucoup plus efficaces, surtout sur les tableaux de large taille. Les tableaux Numpy sont au cœur de presque tout l'écosystème de data science en Python.

Contrairement aux listes en Python, les tableaux Numpy **ne peuvent contenir des membres que d'un seul type**.

Ce type est automatiquement déduit au moment de la création du tableau, et a un impact sur les opérations qui y seront appliquées. On peut aussi spécifier le type manuellement. Pour créer un tableau en python, nous allons utiliser le mot clé **array().**

On peut créer des tableaux de différentes façons dans Numpy

Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Numpy

On peut créer des tableaux de différentes façons dans Numpy:

Créer un tableau depuis une liste Python

```python
# Tableau d'entiers:
np.array([1, 4, 2, 5, 3])
```

Si dans la liste de départ, il y a des données de **types différents**, Numpy essaiera de les **convertir toutes au type le plus général**. Par exemple, les entiers (int) seront convertis en nombres à virgule flottante (float) :

```python
np.array([3.14, 4, 2, 3])
```

```
array([ 3.14, 4. , 2. , 3. ])
```

Il est possible aussi de spécifier le type du tableau :

```python
np.array([1, 2, 3, 4], dtype='float32')
```

Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Numpy

Créer les tableaux directement

Il est souvent plus efficace, surtout pour les tableaux larges, de les créer directement. Numpy contient plusieurs fonctions pour cette tâche.

```python
# Un tableau de longueur 10, rempli d'entiers qui valent 0
np.zeros(10, dtype=int)

# Un tableau de taille 3x5 rempli de nombres à virgule flottante de valeur 1
np.ones((3, 5), dtype=float)

# Un tableau 3x5 rempli de 3,14
np.full((3, 5), 3.14)

# Un tableau rempli d'une séquence linéaire
# commençant à 0 et qui se termine à 20, avec un pas de 2
np.arange(0, 20, 2)

# Un tableau de 5 valeurs, espacées uniformément entre 0 et 1
np.linspace(0, 1, 5)

# Celle-ci vous la conaissez déjà! Essayez aussi "randint" et "normal"
np.random.random((3, 3))

# La matrice identité de taille 3x3
# (matrice identité : https://fr.wikipedia.org/wiki/Matrice_identit%C3%A9)
np.eye(3)
```

Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Numpy

Accès aux éléments du tableau Numpy

Accéder à un seul élément

```python
print(x1)

# Pour accéder au premier élément
print(x1[0])

# Pour accéder au dernier élément
print(x1[-1])

x2 = np.random.randint(10, size=(3, 4))  # Tableau de dimension 2
print(x2[0,1])

# On peut aussi modifier les valeurs
x1[1] = "1000"
print(x1)

# Attention au type
x1[1] = 3.14
print(x1)
```

Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Numpy

Accès aux éléments du tableau Numpy

Accéder à plusieurs éléments

De la même façon que nous pouvons indexer des éléments grâce à **[]**, nous pouvons accéder à un ensemble d'éléments en combinant **[] et :.** La syntaxe suit une règle simple : **x[début:fin:pas].**

NB: Le début peut être omis si on veut commencer au début de la liste (c'est à dire si début = 0). La fin peut être omise si on veut aller jusqu'au bout de la liste (c'est à dire fin = -1 ou fin = len(liste)). Le pas, ainsi que le dernier :, peuvent être omis si le pas est de 1 (-1 si la fin est inférieure au début).

python

```python
print(x1[:5])  # Les cinq premiers éléments

print(x1[5:])  # Les éléments à partir de l'index 5

print(x1[::2])  # Un élément sur deux
```

Si le pas est **négatif**, le début et la fin du slice sont inversés. On peut utiliser cette propriété pour inverser un tableau.

Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Numpy

Quelques fonctions mathématiques avec NumPy

Fonctions trigonométriques

numpy.sin(x)	sinus
numpy.cos(x)	cosinus
numpy.tan(x)	tangente
numpy.arcsin(x)	arcsinus
numpy.arccos(x)	arccosinus
numpy.arctan(x)	arctangente

Fonctions diverses

x**n	x à la puissance n, exemple : x**2
numpy.sqrt(x)	racine carrée
numpy.exp(x)	exponentielle
numpy.log(x)	logarithme népérien
numpy.abs(x)	valeur absolue
numpy.sign(x)	signe

Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Matplotlib

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous formes de graphiques.

Matplotlib utilise le module **pyplot** pour tracer des courbes. Pour l'utiliser, il faut importer les bibliothèques **numpy** et **Matplotlib**.

Utilisation de plot()

L'instruction **plot()** permet de tracer des courbes qui relient des points dont les abscisses et ordonnées sont fournies dans des tableaux.

Exemple :

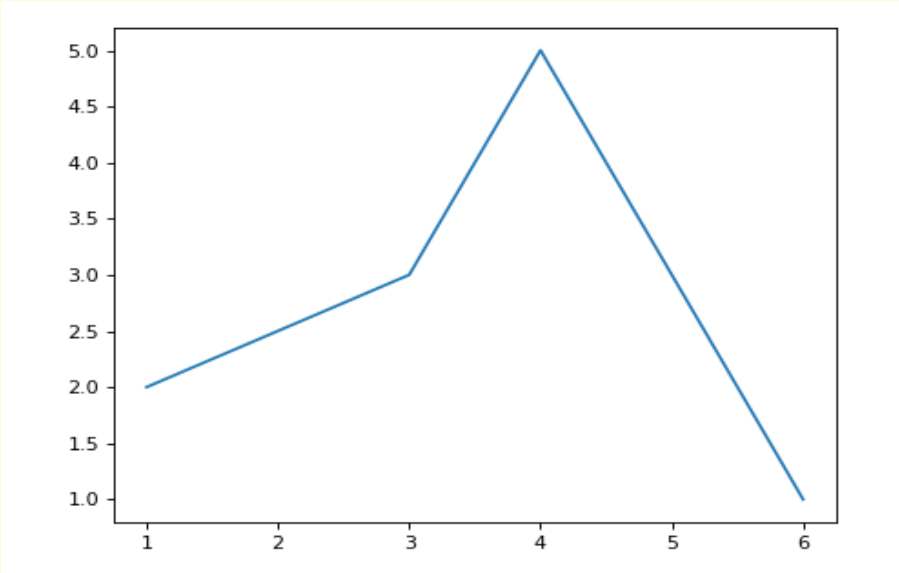
```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1, 3, 4, 6])
y = np.array([2, 3, 5, 1])
plt.plot(x, y)
plt.show() # affiche la figure a l'ecran
```

Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Matplotlib

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1, 3, 4, 6])
y = np.array([2, 3, 5, 1])
plt.plot(x, y)

plt.show() # affiche la figure a l'ecran
```

Résultat :



Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Matplotlib

A tester en exercice

python

```python
# Chanegr la taille de police par défaut
plt.rcParams.update({'font.size': 15})

fig = plt.figure()
ax = plt.axes()
# Couleur spécifiée par son nom, ligne solide
plt.plot(x, np.sin(x - 0), color='blue', linestyle='solid', label='bleu')
# Nom court pour la couleur, ligne avec des traits
plt.plot(x, np.sin(x - 1), color='g', linestyle='dashed', label='vert')
# Valeur de gris entre 0 et 1, des traits et des points
plt.plot(x, np.sin(x - 2), color='0.75', linestyle='dashdot', label='gris')
# Couleur spécifié en RGB, avec des points
plt.plot(x, np.sin(x - 3), color='#FF0000', linestyle='dotted', label='rouge')

# Les limites des axes, essayez aussi les arguments 'tight' et 'equal'
# pour voir leur effet
plt.axis([-1, 11, -1.5, 1.5]);

# Les labels
plt.title("Un exemple de graphe")

# La légende est générée à partir de l'argument label de la fonctio
# plot. L'argument loc spécifie le placement de la légende
plt.legend(loc='lower left');

# Titres des axes
ax = ax.set(xlabel='x', ylabel='sin(x)')
```

Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Seaborn

Seaborn est une librairie qui vient s'ajouter à Matplotlib, remplace certains réglages par défaut et fonctions, et lui ajoute de nouvelles fonctionnalités. Seaborn vient corriger trois défauts de Matplotlib:

- Matplotlib, surtout dans les versions avant la 2.0, ne génère pas des graphiques d'une grande qualité esthétique.
- Matplotlib ne possède pas de fonctions permettant de créer facilement des analyses statistiques sophistiquées.
- Les fonctions de Matplotlib ne sont pas faites pour interagir avec les Dataframes de Panda (que nous verrons au chapitre suivant).

Seaborn fournit une interface qui permet de palier ces problèmes. Il utilise toujours Matplotlib "sous le capot", mais le fait en exposant des fonctions plus intuitives

Seaborn nous fournit aussi des fonctions pour des graphiques utiles pour l'analyse statistique. Par exemple, la fonction **distplot** permet non seulement de visualiser l'histogramme d'un échantillon, mais aussi d'estimer la distribution dont l'échantillon est issu.Il prend en **paramètre un tableau**

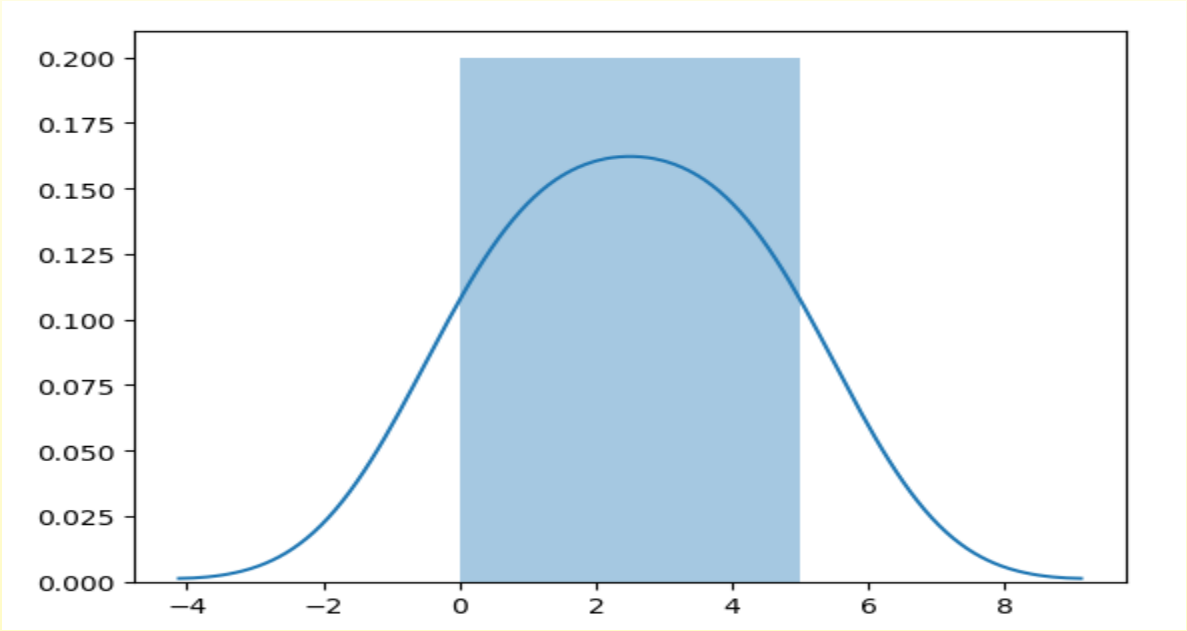
Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Seaborn

Exemple :

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5])

plt.show()
```

Résultat :



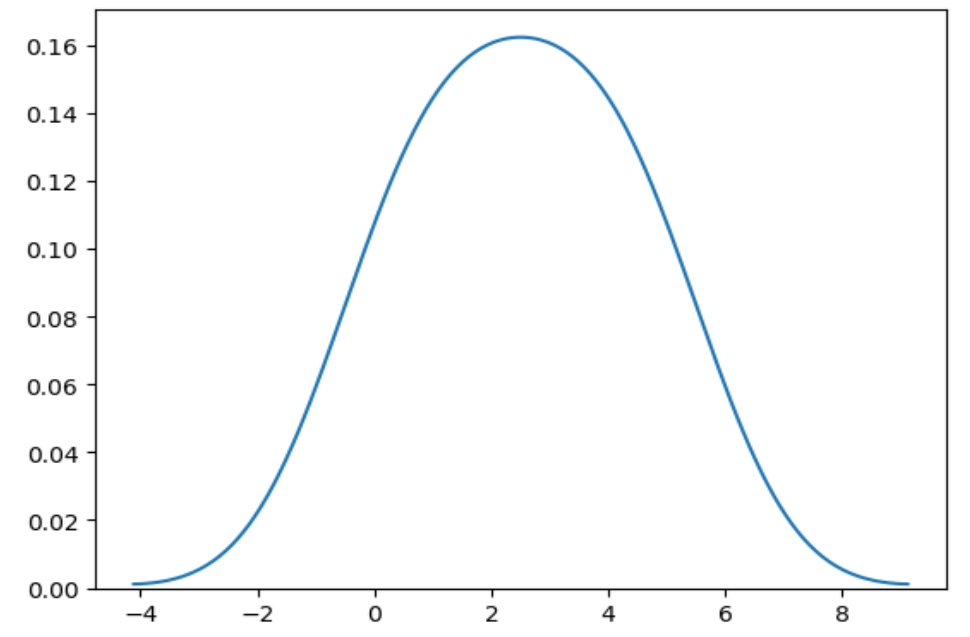
Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Seaborn

Exemple : sans l'histogramme

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5], hist=False)

plt.show()
```

Résultat :



Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Pandas

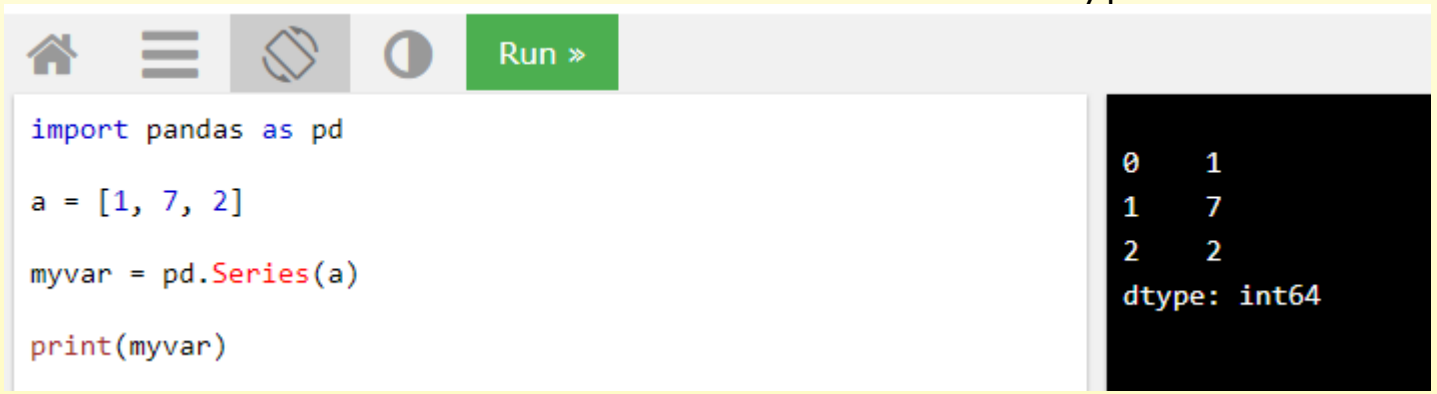
Avec Numpy et Matplotlib , la librairie Pandas fait partie des librairies de base pour la data science en Python. Pandas fournit des structures de données puissantes et simples à utiliser, ainsi que les moyens d'opérer rapidement des opérations sur ces structures. Dans ce chapitre, nous verrons l'intérêt de la librairie Pandas, ainsi que les opérations basiques sur l'objet phare de cette librairie, le dataframe.

Il a des fonctions pour analyser, nettoyer, explorer et manipuler les données.

Pandas series

Une série Pandas est comme une colonne dans un tableau. C'est un tableau unidimensionnel contenant des données de tout type.

Exemple :



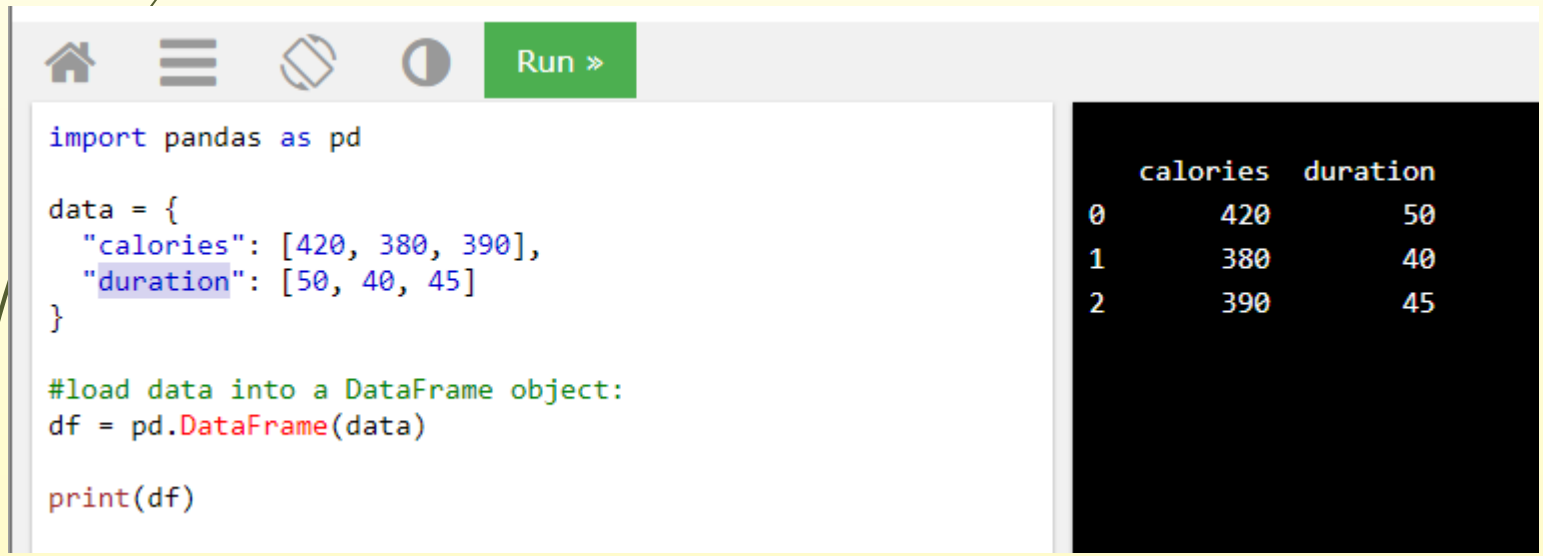
Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Pandas

Pandas DataFrame

Un Pandas DataFrame est une structure de données à 2 dimensions, comme un tableau à 2 dimensions, ou une table avec des lignes et des colonnes.

Exemple :

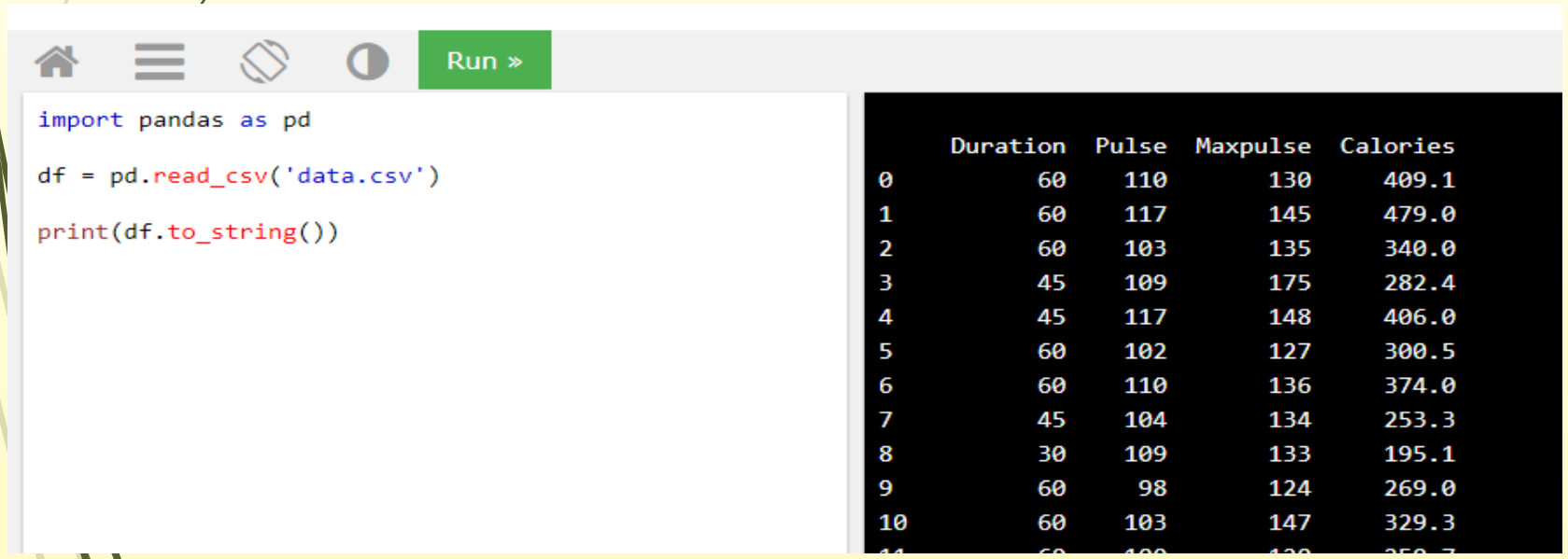


Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Pandas

Pandas Read CSV

Un moyen simple de stocker des données volumineuses consiste à utiliser des fichiers CSV (fichiers séparés par des virgules).Les fichiers CSV contiennent du texte brut et sont un format bien connu qui peut être lu par tout le monde, y compris les **Pandas**. Dans l'exemple ci-dessous, nous utiliserons un fichier CSV appelé «data.csv»: le fichier sera mis à votre disposition.



Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Pandas

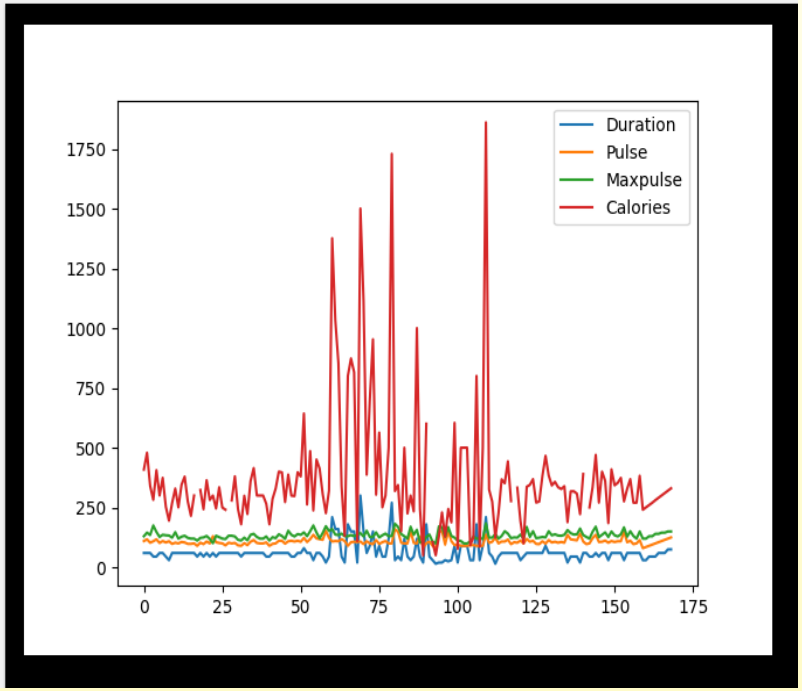
Pandas : Méthode plot ()

Pandas utilise la méthode plot () pour créer des diagrammes. Python utilise Pyplot, un sous-module de la bibliothèque Matplotlib pour visualiser le diagramme à l'écran.

Exemple :

Run »

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot()

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



Unité 6 : Modules et librairies Python pour l'analyse de données

Le package Pandas

Pandas : Méthode plot ()

La méthode plot() peut prendre des arguments. Par exemple , l'argument **kind** permet de définir le type de graphique que l'on souhaite avoir. Il peut prendre des valeur comme **hist** ou **scatter.**

hist : pour un graphique en **histogramme** et **scatter** pour un graphique en **dispersion**.

Unité 7 : Python et les bases de données

In [17]:
```
import mysql.connector as MC

try:
    conn=MC.connect(host='localhost' , unix_socket='/Applications/MAMP/tmp/mysql/mysql.sock
    cursor=conn.cursor()

    req='SELECT * FROM ETUDIANT'
    cursor.execute(req)

    listEtu = cursor.fetchall()

    for i in listEtu:
        #print('Prénom : {}'.format(listEtu[0][1]))
        print(i[1])
except MC.Error as err:
    print(err)
finally:
    if(conn.is_connected()):
        cursor.close()
        conn.close()
```

```
SONTIE
OUEDRAOGO
KEDALO
GOUMBANE
```

Unité 7 : Python et les bases de données


```
import mysql.connector as MC


try:
   conn=MC.connect(host=localhost , unix_socket=/Applications/MAMP/tmp/mysql/mysql.sock,port=8890, database=masterimsd, user=root, password=root)
   cursor=conn.cursor()


   req=SELECT * FROM ETUDIANT
   cursor.execute(req)


   listEtu = cursor.fetchall()


   for i in listEtu:
      #print(Prénom : {}.format(listEtu[0][1]))
```

Conclusion