

Python - bases du langage

lundi 26 septembre 2022 11:57

Lien du cours : <https://openclassrooms.com/fr/courses/7168871-apprenez-les-bases-du-langage-python>

Installez Python sur votre ordinateur

L'installation de Python est très simple ! Rendez-vous sur python.org, choisissez votre système d'exploitation (Mac/Windows/etc.) et cliquez sur le bouton de téléchargement pour installer Python sur votre ordinateur.

Exécutez votre premier programme Python

Maintenant que vous avez installé Python, préparez vous à exécuter votre **premier code Python** !

- Ouvrez Terminal si vous êtes sur macOS ou sur l'invite de commandes si vous êtes sous Windows.
- Écrivez `python` et vous devriez voir des informations à propos de Python apparaître avec `>>>` indiquant où écrire le code.
- Ensuite, écrivez `print("hello, world!")`.

```
PS C:\Users\aresv> python
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("hello, world")
hello, world
```

- Appuyez sur Entrée pour voir le résultat.
- Pour quitter Python, il suffit d'exécuter la commande `exit()`

```
>>> exit()
PS C:\Users\aresv>
```

Ça y est ! Vous avez exécuté votre premier code Python ! Nous avons ouvert un terminal, démarré une console Python (ou Shell) et écrit "hello, world" sur la console.

Comme vous l'avez vu, vous pouvez exécuter chaque ligne de code Python directement dans le **terminal**. Vous pouvez aussi écrire du code dans un **éditeur de texte** séparé puis l'exécuter en utilisant les commandes du terminal. Vous pouvez utiliser de nombreux éditeurs de texte, mais vous pouvez commencer avec Sublime Text (ressource en anglais).

Personnellement, j'utilise VSCode, le meilleur éditeur de texte (cet avis est absolument subjectif)

Voici comment écrire du code dans un éditeur puis l'exécuter :

- Dans l'éditeur, écrivez `print("Hello, world")` .
- Sauvegardez le fichier (par exemple, sous le nom `helloworld.py`).
- Ouvrez Terminal si vous êtes sur macOS ou sur l'invite de commandes si vous êtes sous Windows.

- Dans le dossier Documents, écrivez "python3" et le nom du fichier que vous avez créé : python3 helloworld.py .
- Appuyez sur Entrée pour voir le résultat: Hello, world .

```
PS C:\Users\aresv\documents\projetOpenClassroom\Python> python3 helloworld.py
Hello, world
```

Exercice proposée dans le cours :



```
workspace $ python3
Python 3.10.2 (main, Feb 22 2022, 00:24:37) [GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print (17+35*2)
87
>>> []
```

En résumé

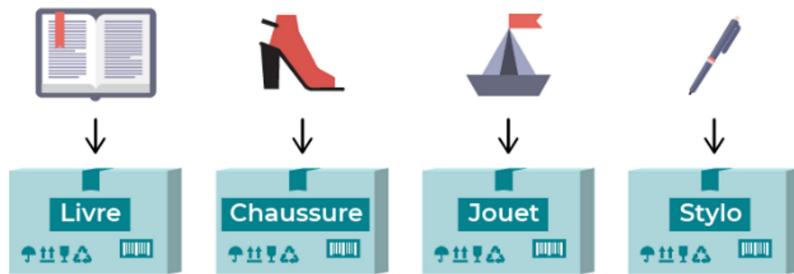
- Python est une langue populaire parmi les débutants et les développeurs experts, car il est facile à lire et polyvalent.
- La partie la plus difficile à propos du codage est de se dépasser pour garder le moral. Vous pouvez le faire !
- Vous pouvez exécuter un script Python dans votre terminal directement à partir d'un éditeur de texte.

Enregistrez vos données avec des variables

Qu'est-ce qu'une variable ?

Imaginons qu'on s'apprête à déménager et qu'il faut ranger nos affaires dans des cartons. Il nous faut un carton pour la vaisselle, un pour les vêtements et un autre pour la télévision. Quand vous faites les cartons, la bonne chose à faire est de leur donner un nom. Ainsi vous saurez ce qui se trouve à l'intérieur.

Une **variable**, c'est comme un **carton**. C'est un **moyen d'enregistrer vos données**. En tant que développeur, il vous faudra enregistrer certaines informations et leur donner un nom afin de pouvoir les récupérer plus tard. Un nom d'utilisateur, des tickets d'avion encore disponibles, le jour de la semaine, un inventaire... Toutes ces données sont enregistrées dans les variables.



Les variables enregistrent les données comme les cartons permettent de ranger les objets.

Il est très facile de déclarer une variable dans Python. Le code ci-dessous permet de déclarer une variable.

```
livre = "Gatsby le Magnifique"
```

Quand on dit **déclarer** (ou **initialiser**) une **variable**, c'est juste une autre façon de dire « **créer une variable** ».

Une **variable** dans **Python** est constituée de **trois éléments** : son **nom**, son **type** et sa **valeur**. Dans ce chapitre nous allons voir le **nom** et la **valeur**, puis nous verrons le type dans le chapitre suivant.

Ici, **livre** est le **nom de la variable** (donc le **nom du carton**) et "**Gatsby le Magnifique**" est la **valeur de la variable** (ce qui se trouve dans le carton). Puis, si vous souhaitez retrouver le titre du livre avec Python, vous pouvez écrire **livre** et vous obtiendrez "**Gatsby le Magnifique**" en retour.

```
type help, copyright, credits
>>> livre = "Gatsby le Magnifique"
>>> livre
'Gatsby le Magnifique'
```

Modifiez une variable

Il est très simple de **modifier** la valeur d'une **variable**. Il suffit simplement d'**assigner une nouvelle valeur à la variable**. Voici l'extrait de code qui **déclare** **livre** avec la valeur "**Gatsby le Magnifique**" puis qui la **modifie** avec "**Beloved**" .

```
>> livre = "Gatsby le Magnifique"
>> print(livre)
Gatsby le Magnifique
>> livre = "Beloved"
>> print(livre)
Beloved
```



Nommez une variable

Un nom de variable doit refléter son contenu comme le nom sur un carton. Voici quelques **recommandations générales** pour choisir un nom :

- **Utilisez des noms descriptifs dans votre code** : Vous avez déjà retrouvé un vieux classeur dans un carton avec le nom « Trucs importants » sur la couverture ? C'était frustrant n'est-ce pas ? Les noms de variables descriptifs et spécifiques simplifient la vie et facilitent la lecture et la modification de votre code. Au lieu de `quantite` (ou pire : `qte`), ajoutez des détails : `quantite_en_stock`, `sold_actuel`, etc.
- **Utilisez des mots complets** : Évitez d'abréger ou de raccourcir les mots autant que possible, même si une alternative plus courte paraît évidente. Par exemple : `revenu_annuel` est **plus clair** que `rev_annuel`.
- **Suivez une convention d'appellation commune** : L'une des conventions d'appellation populaires pour Python est le **snake case** : des noms composés de **plusieurs mots séparés par des tirets bas(_)** comme : `nombre_de_chats`, `reponse_finale` , `le_meilleur Developpeur_python_du_monde`, etc.
- **Commencez avec une lettre ou le tiret bas** : Un nom de variable ne peut pas commencer par un nombre.
- **Utilisez uniquement des caractères alphanumériques et des tirets bas... et donc pas d'accents** : Par exemple, écrivez `bonjour_1` mais pas `bonjour_#1` .
- **N'oubliez pas que les noms de variables sont sensibles à la casse** : `age`, `Age` et `AGE` sont **trois variables différentes**.

Exercice proposé



En résumé

- Une **variable** a un **nom** et une **valeur**, comme un carton avec un nom et des choses à l'intérieur.
- Vous pouvez **déclarer une variable** avec le **nom de la variable** (par exemple, `livre`), un **signe** « égal à » = et la **valeur** (par exemple, "Gatsby le Magnifique"). Il se peut que la **valeur nécessite des guillemets avant et après**, mais cela **dépend du type de donnée dont il s'agit** (ce que nous traiterons dans le chapitre suivant).
- N'oubliez pas ces **bonnes pratiques** quand vous **nommerez une variable**.

Classez des données avec les types de données

Qu'est-ce qu'un type de données ?

Un **type de données** est simplement une façon de classer différents **types de variables**. Quand vous avez un fruit, vous pouvez le classer dans un type spécifique comme pomme ou orange par exemple. Quand vous avez une variable, l'ordinateur doit la classer dans un type spécifique aussi : un type de données. C'est comme les types de données que vous pouvez utiliser dans un tableur Excel : les nombres, les mots, etc.

Les types de données les plus simples, ou **primitifs**, utilisés dans Python sont :

- Les **entiers** (**Integers** en anglais)
- Les **virgules flottantes** (**Float** en anglais)
- Les **chaînes de caractères** (**Strings** en anglais)
- Les **booléens**

Découvrez les types de données numériques : entiers et virgules flottantes

Les **types numériques** se divisent en **deux catégories** dans **Python** : les **entiers** (**Integers** en anglais) et les **virgules flottantes** (**Float** en anglais).

Un **entier** est un **nombre entier** :

- **1**
- **4**
- **3 934**

Une **virgule flottante** est un **nombre décimal** :

- **3,14**
- **563,2**
- **99,9**

Les nombres **100** et **100,0** ont l'air de se ressembler, mais dans Python, **l'un est un entier et l'autre est une virgule flottante**. Vous devinez lequel est lequel ? (Astuce : **celui qui a une décimale est une virgule flottante !**)

Tout comme les nombres dans Excel ou avec une calculatrice, vous pouvez **réaliser toutes sortes d'opérations arithmétiques** avec des **entiers** et des **virgules flottantes** dans Python. Voici quelques exemples d'**opérateurs arithmétiques** que vous pouvez utiliser dans votre code :

- **x + y** : la **somme** de x et y (x plus y).
- **x - y** : la **différence** entre x et y (x moins y).
- **x * y** : le **produit** de x et y (x fois y).
- **x / y** : le **quotient** de x et y (x divisé par y).
- **x % y** : le **reste** de x divisé par y.

Expérimenter avec l'arithmétique est un excellent moyen de s'essayer à un peu de programmation basique.

Définissez les chaînes de caractères

Une **chaîne de caractères**, c'est juste un mot sophistiqué pour ne pas dire... et bien, **mots**. Une **chaîne** c'est une **information (généralement du texte)** entourée de guillemets (soit **simples** soit **doubles**). Donc '**bonjour**' avec des **guillemets simples** et "**bonjour**" avec des **guillemets doubles** sont tous les deux des chaînes. Et "**au revoir**" et "**quoi de neuf**" aussi.

Quand on utilise des nombres dans une variable, il faut juste se rappeler que '**912**' et "**912**" sont des **chaînes** simplement parce qu'ils sont entourés par des **guillemets**, alors que **912** est un **nombre entier**. Concrètement, comme ce sont des informations dans votre code, '**912**' et "**912**" fonctionnent en tant que **texte**, alors que **912** fonctionne en tant que **valeur numérique**.

Voici quelques autres exemples de variables de type **string (chaîne de caractères)** :

- **salutations** = "bonjour !"
- **le_professeur** = "Paul-Emile"
- **prenom** = "Sam"
- **citation_inspirante** = "vous ratez 100 % des opportunités que vous ne saisissez pas"

Les booléens : vrai ou faux ?

Ne soyez pas intimidé par ce nom étrange ! Un **booléen** c'est assez simple. Il n'y a que deux options de valeur : **True** (**vrai**) ou **False** (**faux**).

Les booléens peuvent être très utiles pour déterminer si une action est réussie ou pas.

Par exemple, si vous voulez **avoir une variable qui enregistre le fait qu'il y a du soleil dehors ou pas**, vous pouvez dire **climat_ensoleille = True**. S'il n'y a pas de soleil, alors **climat_ensoleille = False**.

Exercice proposé

Tasks		
<input checked="" type="checkbox"/>	Ecrivez votre nom dans une variable appelée <code>nom</code> .	>
<input type="checkbox"/>	Utilisez la fonction <code>print</code> pour afficher le contenu de la variable <code>nom</code> dans le terminal.	>
<input checked="" type="checkbox"/>	Stockez le résultat de <code>x + y</code> dans une variable notée <code>z</code> .	>
<input type="checkbox"/>	Affichez le résultat de la variable <code>z</code> dans le terminal à l'aide de la fonction <code>print</code>	>

```
main.py
1 nom = "Vel"
2 x = 21
3 y = 21
4 z = x+y
5
6 print (nom)
7 print ([z])
8
```

```
workspace $ python3 main.py
Vel
42
workspace $
```

En résumé

- Un **type de données** est une **catégorie** ou une **classe d'information** dans votre code.
- Il y a de **nombreux types de données**, mais les plus simples sont les **virgules flottantes (nombres décimaux)**, les **entiers (nombres entiers)**, les **booléens (vrai/faux)** et les **chaînes de caractères (texte entre guillemets)**.

Enregistrez des groupes de données avec les listes

Qu'est-ce qu'une liste et pourquoi l'utiliser ?

Les pommes, les oranges, les poires. 🍎🍊🍐 Les chiens, les chats, les lapins. 🐕🐩🐰 Vous utilisez les **listes** tout le temps dans votre vie quotidienne et c'est pareil dans Python. Vous devez avoir une **liste** pour y enregistrer une **collection d'objets** auxquels vous voulez accéder plus tard.

Dans Python, on utilise des **crochets []** pour indiquer une liste. Le **code suivant crée une liste de différentes plateformes de réseaux sociaux et la sauvegarde dans une variable** appelée **plateformes_sociales**.

- `plateformes_sociales = ["Facebook", "Instagram", "Snapchat", "Twitter"]`

Les listes acceptent n'importe quel type de données aussi, et même des combinaisons de types de données.

Accédez aux éléments d'une liste

Pour accéder aux **éléments d'une liste**, on utilise un **indice**. Chaque élément a un indice qui lui correspond, selon sa position dans la liste. Vous obtenez la valeur de cet indice avec la syntaxe suivante : `liste[indice]`. Elle vous **renverra la valeur de la liste qui est à la position de l'indice**.

L'important est de noter que dans la plupart des langages de programmation, y compris Python, l'**indice commence à 0, pas 1**. Donc si vous voulez accéder au premier élément dans la liste ci-dessus, vous devez taper :

- >> plateformes_sociales[0]
"Facebook"

Pour accéder au deuxième élément, tapez :

- >> plateformes_sociales[1]
"Instagram"

et ainsi de suite.

En particulier dans Python, vous pouvez aussi accéder aux éléments en sens inverse, en utilisant des **nombres négatifs**. Pour accéder au **dernier élément de la liste**, utilisez l'**indice -1**.

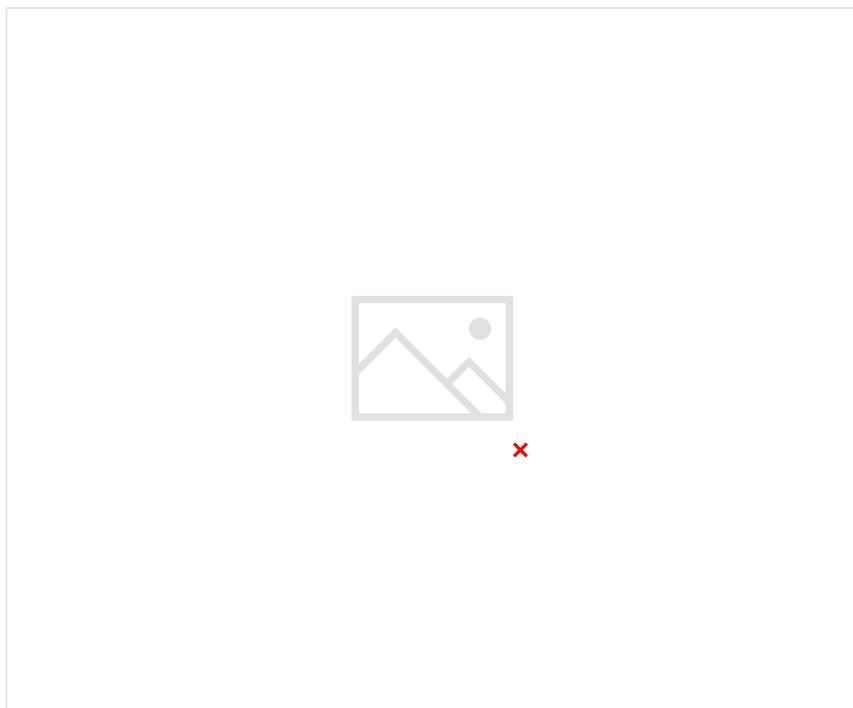
- >> plateformes_sociales[-1]
"Twitter"

Accédez aux caractères d'une chaîne comme un élément d'une liste

Les **indices** fonctionnent aussi avec les **chaînes de caractères** ! En fait, les chaînes de caractères sont juste des listes de caractères. Chaque **caractère** correspond à un **indice** qui va de **zéro à la longueur de la chaîne**.

Par exemple, dans la chaîne `langage = "PYTHON"` , `langage[2]` vous renverra "`T`". Tout simplement parce que l'**indice 2** dans le mot "`PYTHON`" est le "`T`". Ou bien, avec l'**indice de la position inverse**, vous devez utiliser `langage[-4]` pour accéder à "`T`".

- >> langage_de_programmation = "PYTHON"
>> langage_de_programmation[2]
"T"
>> langage_de_programmation[-4]
"T"



Modifiez, retirez et triez les listes

Dans Python, c'est très simple de réaliser toutes sortes d'**opérations avec les listes**. À la place de créer une toute nouvelle liste à chaque fois que vous voulez **ajouter, retirer ou trier des éléments**, vous pouvez **faire appel à une méthode de liste**. Nous allons voir les méthodes plus tard, mais pour l'instant, vous devez juste savoir qu'**une méthode est une façon de réaliser une opération spécifique sur un élément**.

Par exemple, pour **ajouter une plateforme de réseau social à la fin de la liste existante**, vous pouvez utiliser la méthode **append()** :

```
- >> plateformes_sociales.append("TikTok")
>> print(plateformes_sociales)
["Facebook", "Instagram", "Snapchat", "Twitter", "TikTok"]
```

Pour **retirer un élément spécifique d'une liste**, vous pouvez utiliser la méthode **remove()**

```
- >> plateformes_sociales.remove("Snapchat")
>> print(plateformes_sociales)
["Facebook", "Instagram", "Twitter", "TikTok"]
```

remove() retire uniquement **la première instance** du terme que vous saisissez.

Pour connaître la **longueur de la liste**, utilisez la méthode **len()**.

```
- >> len(plateformes_sociales)
4
```

La dernière méthode que nous allons voir est **sort()**. Elle **trie les éléments de la liste**. Le tri se fait **alphabétiquement** pour les **listes de chaînes** et **numériquement** pour les **listes de nombres**.

```
- >> plateformes_sociales.sort()
```

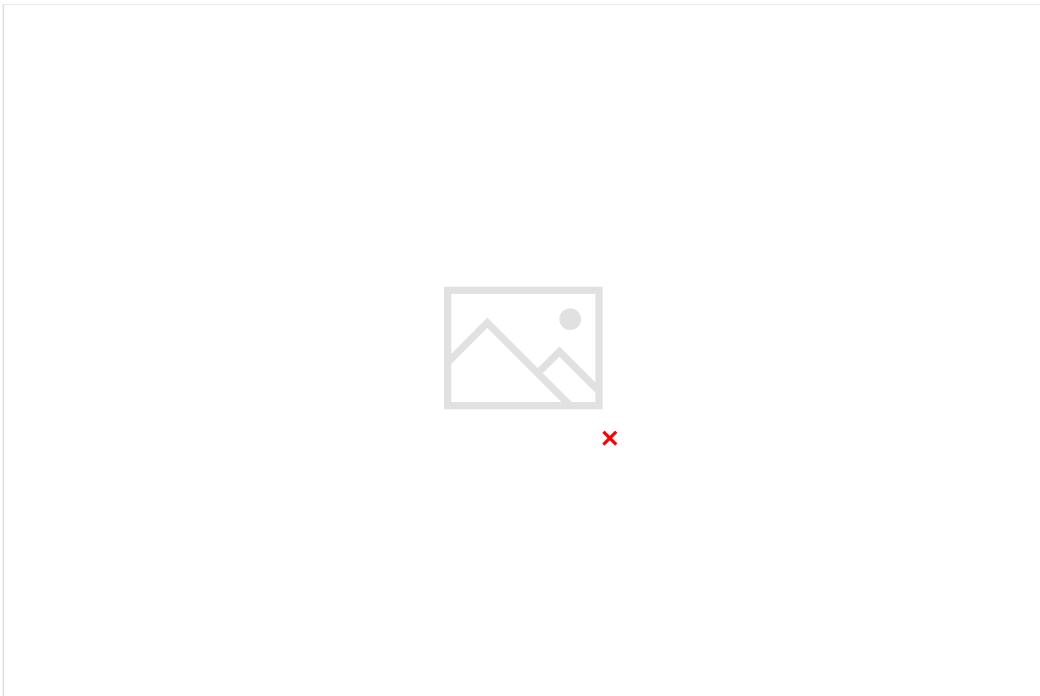
Découvrez les tuples

Les **tuples** sont une autre structure de Python qui **sert à enregistrer des données**. Au lieu des **crochets []**, ils se caractérisent par les **parenthèses ()**.

```
- plateformes_sociales_tuple = ("Facebook", "Instagram", "TikTok", "Twitter")
```

Beaucoup des **propriétés des tuples sont similaires à celles des listes normales**. Par exemple, les **listes et tuples utilisent tous deux les indices**. La principale différence est que les **tuples sont immuables (ne peuvent pas être modifiés)**, alors que les **listes sont modifiables**.

Exercice :



En résumé

- Une liste est un moyen d'enregistrer plusieurs données ensemble.
- Accédez aux éléments de la liste via l'indice, qui commence à 0.
- Vous pouvez ajouter, retirer, trier les listes et bien plus en utilisant les méthodes de listes.
- Les tuples sont comme des listes, mais sont définis par des parenthèses () à la place de crochets [] et sont immuables.

Enregistrez des données complexes avec des dictionnaires

Qu'est-ce qu'un dictionnaire ?

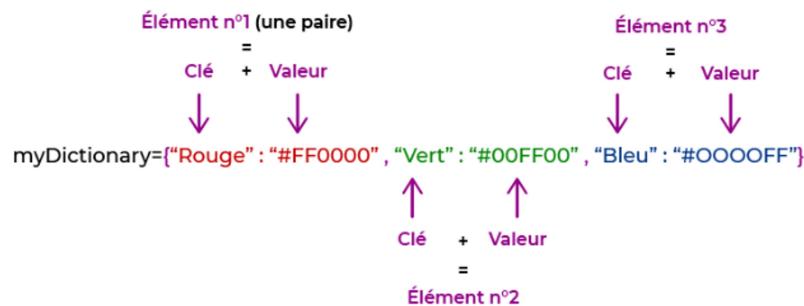
Parfois vous avez besoin de représenter des **données plus complexes** que des **nombres**, des **chaînes** ou des **listes**.

Un **dictionnaire** est une structure de données qui enregistre des données dans des paires clés-valeurs. Voici un exemple d'une **clé** et d'une **valeur** : **responsable_de_campagne: "Jeanne d'Arc"** . La clé est "**responsable_de_campagne**" et la valeur est "**Jeanne d'Arc**".

Les **dictionnaires** sont indiqués par des **accolades {}** au **début** et à la **fin**. Chaque paire **clé-valeur** comprend un **deux-**

points : placé entre la clé et la valeur et une virgule , à la fin. Chaque dictionnaire doit être composé de clés uniques.

Dans le diagramme ci-dessous, le **dictionnaire défini a trois éléments et chaque élément est une paire clé-valeur**.



Chaque élément dans le dictionnaire est une paire clé-valeur.

Créez un dictionnaire

Imaginons que vous voulez enregistrer des informations à propos d'une nouvelle campagne pour une entreprise de nourriture pour chien. Vous allez probablement devoir **sauvegarder un nom de campagne, des dates de début et de fin, le nom d'un responsable de campagne et des influenceurs importants**. Vous pouvez **sauvegarder toutes ces informations dans une variable avec un dictionnaire**.

Pour enregistrer tout cela, vous pouvez sauvegarder un **dictionnaire** comme ça :

```
nouvelle_campagne = {  
    "responsable_de_campagne": "Jeanne d'Arc",  
    "nom_de_campagne": "Campagne nous aimons les chiens",  
    "date_de_début": "01/01/2020",  
    "influenceurs_importants": ["@MonAmourDeChien", "@MeilleuresFriandisesPourChiens"]  
}
```

Vous pouvez aussi **créer un nouveau dictionnaire avec des accolades vides {} ou la fonction dict()** et avec des **paires clés-valeurs** comme indiqué ci-dessous :

```
taux_de_conversion = {}  
taux_de_conversion['facebook'] = 3.4  
taux_de_conversion['instagram'] = 1.2  
taux_de_conversion = dict()  
taux_de_conversion['facebook'] = 3.4  
taux_de_conversion['instagram'] = 1.2
```

Accédez à une valeur dans un dictionnaire

Pour accéder aux **différentes valeurs**, vous pouvez utiliser la **clé** pour chacune des paires **clés-valeurs**.

```
>> nouvelle_campagne['responsable_de_campagne']  
"Jeanne d'Arc"
```

```
>> taux_de_conversion['facebook']
3.4
```

Réalisez des opérations courantes avec les dictionnaires

Comme pour les listes, **plusieurs méthodes (ou opérations) intégrées à Python vous permettent de manipuler les données dans les dictionnaires.**

Ajoutez une paire clé-valeur

Pour ajouter une paire clé-valeur à un dictionnaire, ajoutez juste une nouvelle clé dans le dictionnaire existant. Si la clé existe déjà, vous l'écraserez en définissant une valeur. Le code suivant crée un dictionnaire appelé **infos_labradoodle** et enregistre des informations à propos du poids et de l'origine des **labradoodles**, un croisement de chiens.

```
infos_labradoodle = {
    "poids": "13 à 16 kg",
    "origine": "États-Unis"
}
```

Pour ajouter une nouvelle clé-valeur comme le nom scientifique du labradoodle, ajoutez simplement :

```
infos_labradoodle['nom_scientifique'] = "Canis lupus familiaris"
```

Maintenant, `infos_labradoodle` renvoie :

```
infos_labradoodle = {
    "poids": "13 à 16 kg",
    "origine": "États-Unis",
    "nom_scientifique": "Canis lupus familiaris"
}
```

Si vous écrivez `infos_labradoodle["poids"] = "45 kg"` , la valeur existante sera écrasée et le résultat sera donc :

```
>> infos_labradoodle["poids"]
"45 kg"
```

Supprimez une clé-valeur

Pour supprimer une paire clé-valeur, vous pouvez utiliser le mot-clé **del** et la clé que vous voulez supprimer ou encore la méthode **pop**. Pour supprimer la paire clé-valeur "origine" de la paire clé-valeur, écrivez :

```
>> del infos_labradoodle["origine"]
>> print(infos_labradoodle)
{ "poids": "13 à 16 kg",
  "nom_scientifique": "Canis lupus familiaris"}
```

Qu'est-ce qu'un mot-clé ?

Certains mots font partie du langage Python et ne peuvent pas être utilisés comme noms de variables. Par

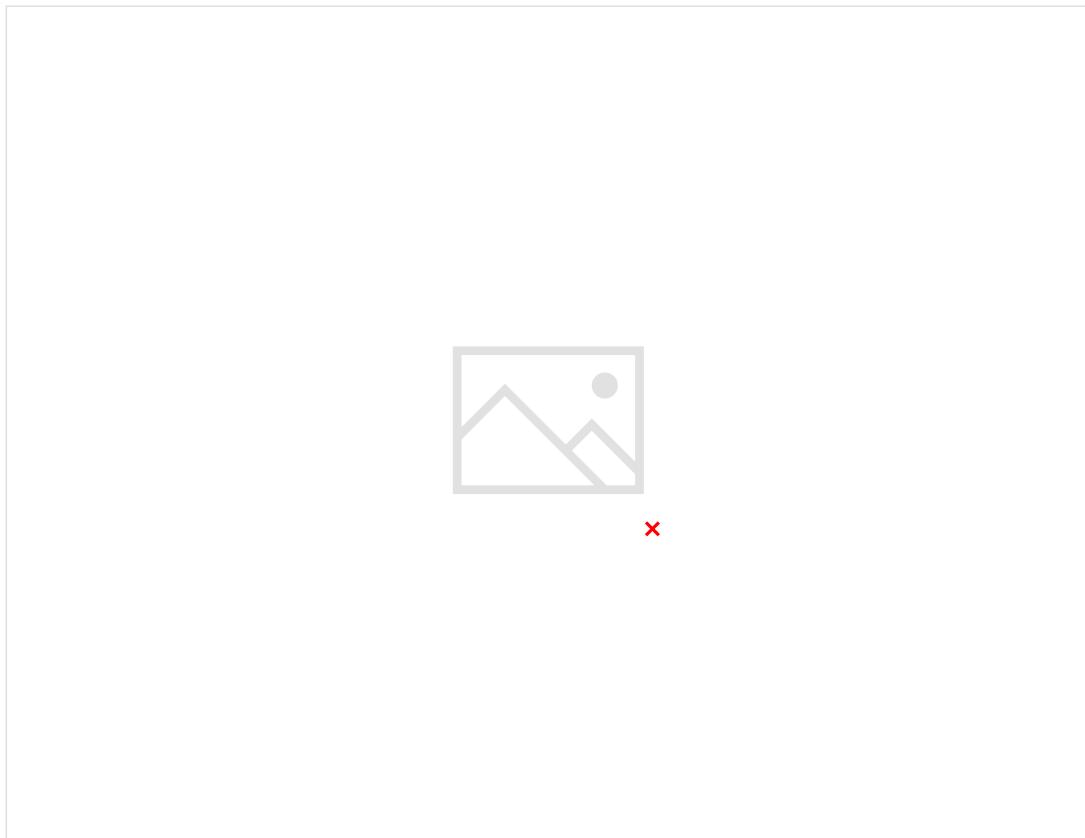
exemple, **del**, **if** et **else**. Ces mots sont connus comme étant des mots réservés ou des mots-clés.

Vérifiez l'existence d'une clé spécifique

Vous pouvez utiliser le mot-clé **in** pour vérifier si une clé spécifique existe dans un dictionnaire. Pour faire cela, spécifiez la clé que vous voulez rechercher, écrivez le mot-clé **in** et le nom de la variable du dictionnaire que vous examinez. Le résultat renvoie un booléen qui indique si la clé est dans ce dictionnaire. Par exemple, si vous voulez voir si la clé « **poids** » existe dans votre dictionnaire **infos_labradoodle** , écrivez le code qui suit :

```
>> "poids" in infos_labradoodle  
True  
>> "race" in infos_labradoodle  
False
```

Exercice proposé :



En résumé

- Un **dictionnaire** est un moyen d'enregistrer des paires clés-valeurs qui représentent un **objet** plus grand.
- Vous pouvez créer un **dictionnaire** avec des accolades {} et y mettre toutes les **clés-valeurs** dès le début ou les ajouter au fur et à mesure.
- **Chaque clé dans un dictionnaire doit être unique.**

[Contrôlez le déroulement de votre programme avec des conditions](#)

Qu'est-ce que le déroulement du programme ?

Le **déroulement du programme** est l'**ordre** dans lequel les **lignes de code** sont **exécutées**. Certaines lignes seront lues une fois seulement, d'autres plusieurs fois. D'autres encore pourraient être complètement ignorées, tout dépend de la façon dont vous les avez codées.

Dans ce premier chapitre sur le déroulement du programme, nous allons regarder comment programmer votre code avec des **instructions conditionnelles**.

Les **instructions conditionnelles** sont un moyen de contrôler la logique et le déroulement de votre code avec des **conditions**. Vous prenez tout le temps des décisions d'après des conditions dans votre vie quotidienne.

Par exemple : S'il fait beau dehors, je vais à la plage. Dans ce scénario, votre décision d'aller à la plage dépend du climat. Le climat est donc la condition.

Voyons comment ce type de logique fonctionne dans le code.

Définissez des conditions avec les instructions if/else

L'un des blocs essentiels à la structure d'un **déroulement conditionnel** est l'instruction **if**.

Avec une instruction **if**, vous pouvez exécuter certaines lignes de code uniquement si une certaine condition est **vraie (True)**. Si cette condition est **fausse (False)**, le code ne s'exécutera pas.

Voici un exemple sur le climat :

```
if ensoleille:  
    print("on va à la plage !")
```

La ligne 2 s'exécute uniquement si la condition **ensoleille** est **True**.

Ça a l'air assez simple, alors quand est-ce que « **else** » entre en jeu ?

Gardons l'exemple du climat et allons plus loin pour y ajouter une instruction « **else** » :

S'il fait beau dehors, je vais à la plage.

Sinon (par exemple s'il pleut), je reste à la maison !

Avec les instructions **if/else** dans Python, la syntaxe ressemble à ça :

```
if mon_booleen:  
    # exécuter le code quand mon_booleen est vrai  
else:  
    # exécuter le code quand mon_booleen est faux
```

Donc pour afficher une instruction sous une certaine condition, vous pouvez suivre les étapes suivantes :

```
ensoleille = True
```

```
if ensoleille:  
    print("on va à la plage !")  
else:  
    print("on reste à la maison !")
```

Dans cet extrait de code, puisque **ensoleille** est **vrai**, vous verrez **on va à la plage !** s'afficher dans votre terminal. Si sa **valeur** était **fausse**, le code afficherait **on reste à la maison !**

Définissez des conditions alternatives en ajoutant une clause **elif**

Les instructions **if/elif/else** vous permettent de définir des **conditions multiples**. Le mot-clé **elif** vous permet d'ajouter autant de **conditions que vous voulez**. Vous devez ensuite terminer avec une instruction **else**.

Disons que vous voulez aller à la plage s'il fait chaud dehors, et que vous voulez faire un bonhomme de neige s'il neige. Et sinon, vous restez à la maison. Vous pouvez coder tout ça avec la syntaxe ci-dessous :

```
ensoleille = False  
neige = True
```

```
if ensoleille:  
    print("on va à la plage !")  
elif neige:  
    print("on fait un bonhomme de neige")  
else:  
    print("on reste à la maison !")
```

Ce code va **vérifier si ensoleille est vrai**, et parce que c'est **faux**, il va **vérifier si neige est vrai**. Comme **neige** est **vrai**, le code va afficher **faire un bonhomme de neige**. Mais si **neige** était **aussi faux**, le **programme aurait exécuté l'instruction finale else et affiché on reste à la maison** !

Définissez des conditions multiples avec des opérateurs logiques

Si vous voulez vérifier **plusieurs conditions** pour un seul résultat dans la même **instruction if**, vous pouvez utiliser les **opérateurs logiques** :

- **and**: vérifie si deux conditions sont toutes les deux vraies.
- **or**: vérifie si au moins une condition est vraie.
- **not**: vérifie si une condition n'est pas vraie (c'est-à-dire fausse).

Ces opérateurs peuvent être **combinés** et **mélangés** selon vos **besoins**.

Admettons que vous voulez aller à la plage seulement s'il y a du soleil et que c'est le weekend. Mais s'il y a du soleil et que nous sommes au milieu de la semaine, vous devez être au travail.

```
avec_soleil = True  
en_semaine = False  
  
if avec_soleil and not en_semaine:  
    print("on va à la plage !")  
elif avec_soleil and en_semaine:  
    print("on va au travail !")  
else:  
    print("on reste à la maison !")
```

Définissez des conditions complexes avec des expressions comparatives

Les **expressions comparatives** vous permettent de comparer différentes expressions entre elles et d'évaluer si une expression est vraie ou fausse.

Si vous avez deux valeurs, a et b, vous pouvez utiliser les opérateurs de comparaison suivants dans Python :

- Égal à : a == b
- Non égal à : a != b
- Moins que : a < b
- Moins que ou égal à : a <= b
- Plus que : a > b
- Plus que ou égal à : a >= b
- Par exemple :

```
nombre_de_sieges = 30
nombre_dinvites = 25

if nombre_dinvites < nombre_de_sieges:
    # autoriser plus d'invités
else:
    # ne pas autoriser plus d'invités
```

Attention à ne pas vous tromper avec le comparateur « égal à » ==. Quand vous testez si une variable est égale à une autre, vous devez utiliser == au lieu de =. **Quand vous voulez assigner une valeur à une variable, vous devez utiliser un « égal à » =.**

Assigner à une variable :

```
nombre_de_sieges = 30
```

Tester si **nombre_de_sieges** est égal à 30 :

```
nombre_de_sieges == 30
```

Remarque rapide sur les **indentations** :

Vous avez vu un format spécifique d'**indentation** dans les exemples de code précédents. Après une instruction **if**, tout le code qui suit doit être **indenté**, c'est-à-dire qu'il doit être décalé par rapport à la marge. Sinon **Python ne fait pas la différence** entre l'**instruction** et le **code à exécuter**.

Exercice : deviner le fonctionnement des conditions :

Exo 1 :

```
1 # 1) Comprenez le code ci-dessous
2 a = True
3 b = True
4 c = False
5
6 if a and b:
7     x = 9
8 elif c:
9     x = 4
10 elif not a:
11     x = 8
12 else:
13     x = 7
14 # 2) Déduisez-en la valeur de x et assignez la dans la variable nommée "solution"
15 solution = 9
16 # 3) Vérifiez votre intuition en faisant tourner le code (> "Run")
17 print(f"{solution} est la bonne valeur de x !" if solution == x else "Raté")
18
```



> Terminal x +

```
workspace $ python3 exercice_2.py
False est la bonne valeur de vrai_ou_faux !
workspace $ []
```

Exo 2 :



En résumé

- Les **instructions if/else** vous aident à définir **certaines conditions** quand le code est exécuté.
- Le mot-clé **elif** vous permet d'utiliser **plusieurs conditions**.

- Vous pouvez grouper **différentes conditions** ensemble avec **and**, **or** et **not**.
- Les **opérateurs de comparaison** comme < **and** > vous permettent de **comparer plusieurs variables**.

Répétez des tâches facilement à l'aide de boucles

Quand utiliser des boucles

En programmation, il y a des ensembles d'**instructions qu'il faut répéter plusieurs fois**. Par exemple, si vous voulez **réaliser une même tâche sur tous les éléments d'une liste**. Et si vous avez une liste de campagnes et que vous voulez les afficher toutes ?

Quand vous avez besoin de répéter un ensemble d'instructions, parfois vous savez combien de fois vous allez le répéter, d'autres fois vous ne savez pas. Parfois, le nombre de répétitions n'est pas important et vous voulez répéter le code jusqu'à ce qu'une certaine condition soit remplie.

Pour tous ces usages, vous allez utiliser des **boucles**.

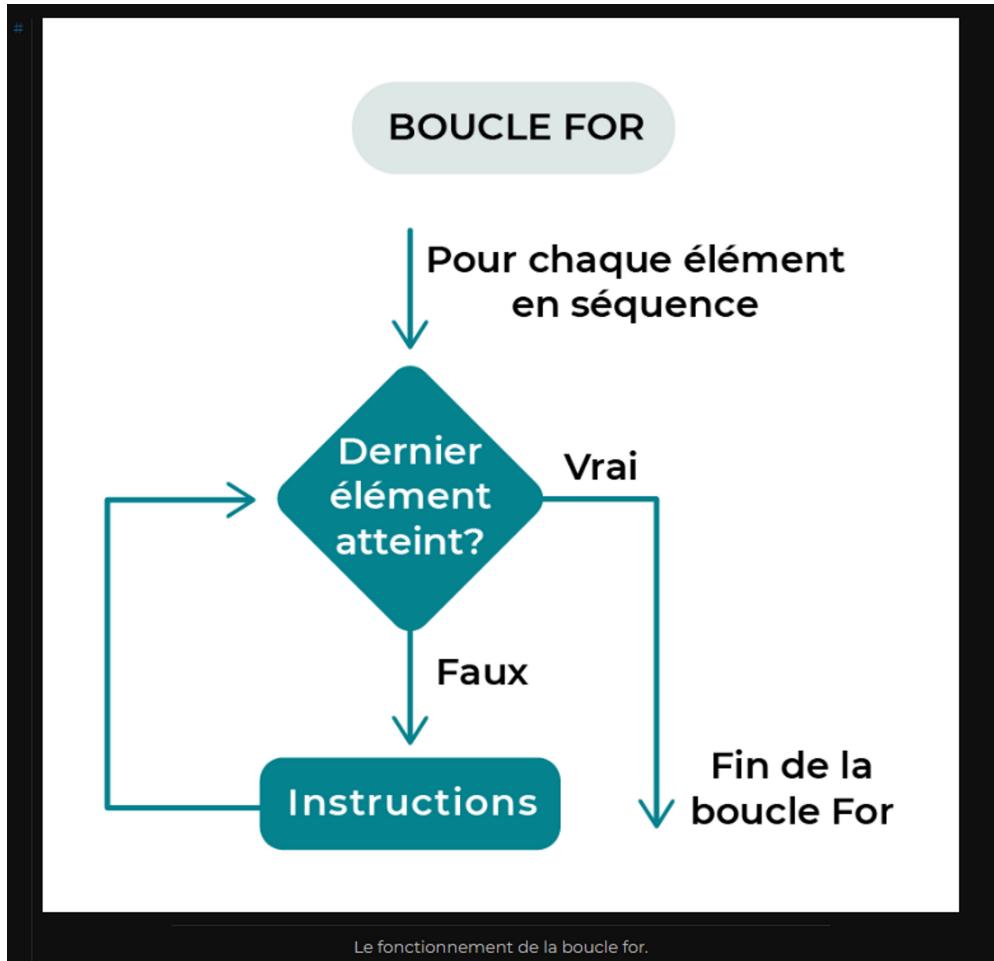
La boucle for

La **boucle for** est le **type de boucle centrale dans Python**. Une boucle **for** est utilisée pour **itérer sur une séquence**. Ça peut être une **liste**, un **tuple**, un **dictionnaire** ou même une **chaîne de caractères**. Avec une **boucle for**, vous pouvez **exécuter le même code pour chaque élément dans cette séquence**.

Avec Python, c'est très facile de créer des **boucles**. Si vous voulez afficher tous les éléments dans une liste, le code ressemblera à ça :

```
races_de_chien = ["golden retriever", "chihuahua", "terrier", "carlin"]  
for chien in races_de_chien:  
    print(chien)
```

Dans ce code, chaque élément dans **races_de_chien** sera affiché dans le terminal. **chien** est un nom de variable qui se met à jour pour être l'élément suivant à chaque fois que la boucle se répète. Vous pouvez modifier **chien** en **cocorico** et il s'exécutera toujours de la même façon, mais cela n'a pas bien d'intérêt !



Vous pouvez faire le même genre de **boucle for** si vous voulez passer en **boucle chaque caractère d'une chaîne**.

Pour **boucler un certain nombre de fois**, vous pouvez utiliser la **fonction range()**. Elle renverra une séquence de nombres qui vont de **0 à un nombre de fin déterminé**.

Vous n'avez pas encore appris les fonctions, mais ça ne va pas tarder ! Pour l'instant, vous devez juste savoir qu'on peut saisir un nombre dans la **fonction range()** et qu'elle renvoie une séquence de nombres allant de 0 à ce nombre moins un.

```
for x in range(5):
    print(x)
```

Ce code affichera 0, 1, 2, 3, 4 en séquence.

```
for x in range(100):
    print(f'{x} bouteilles de bières au mur !')
```

Les accolades **{}** ci-dessus prennent n'importe quelle valeur dans la **variable x** et la remplace (n'oubliez pas le **"f"** au début de la **string** qui signifie **format**). Donc dans cet exemple, le code va afficher :

```
0 bouteilles de bières au mur !
1 bouteilles de bières au mur !
...
99 bouteilles de bières au mur !
```

```
🐍 boucleFor.py > ...
1   for x in range(11):
2     |   print (f"{x} Lalafell écrasé par un Moai")

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL CODEWHISPERER RE
PS C:\Users\aresv\Documents\projetOpenClassroom\Python> & C:/Users, 
sers/aresv/Documents/projetOpenClassroom/Python/boucleFor.py
0 Lalafell écrasé par un Moai
1 Lalafell écrasé par un Moai
2 Lalafell écrasé par un Moai
3 Lalafell écrasé par un Moai
4 Lalafell écrasé par un Moai
5 Lalafell écrasé par un Moai
6 Lalafell écrasé par un Moai
7 Lalafell écrasé par un Moai
8 Lalafell écrasé par un Moai
9 Lalafell écrasé par un Moai
10 Lalafell écrasé par un Moai
PS C:\Users\aresv\Documents\projetOpenClassroom\Python> []
```

La boucle while

La **boucle for** vous permet d'**exécuter du code un nombre spécifique de fois**, alors que la **boucle while** continue de s'**exécuter jusqu'à ce qu'une certaine condition soit remplie**.

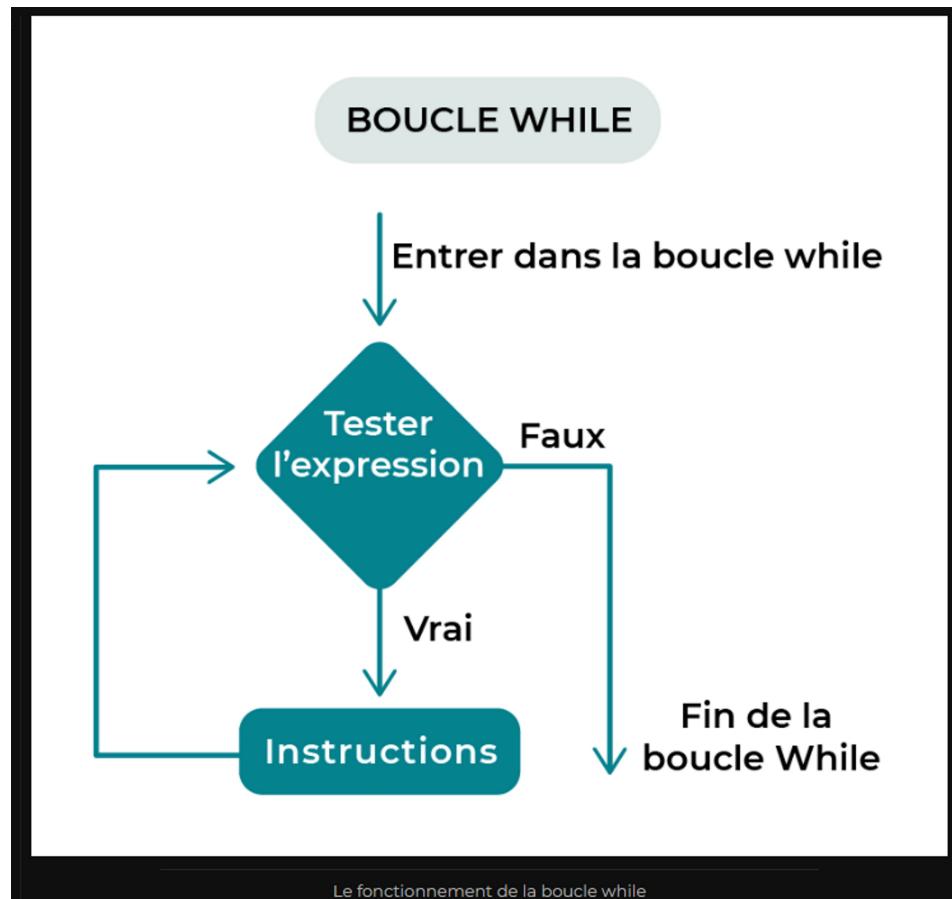
Dans le chapitre précédent, vous avez découvert des **conditions**, ou **instructions**, qui évaluent si une expression est **vraie** ou **fausse**. C'est la même chose ici : le code dans l'instruction **while** s'**exécute jusqu'à ce que la condition devienne fausse**.

L'extrait de code ci-dessous vérifie la capacité actuelle et l'augmente d'une unité jusqu'à ce que la capacité maximale soit atteinte (+= 1augmente la valeur actuelle de 1).

```
capacite_maximale = 10
capacite_actuelle = 3
while capacite_actuelle < capacite_maximale:
    capacite_actuelle += 1
```

Comme cette **capacite_actuelle** commence à 3, ce code s'exécute 7 fois jusqu'à ce que **capacite_actuelle** atteigne 10.

```
🐍 boucleWhile.py > ...
1   capacite_max = 10
2   chat_actuel = 0
3   while chat_actuel < capacite_max:
4     |   chat_actuel +=1
5     |   print (f"{chat_actuel} Lalafell sont devenue Miqo'te")
```



Le fonctionnement de la boucle while

C'est important de connaître les **boucles infinies**. Si la **condition** que vous avez réglée est toujours **vraie**, la **boucle** va **s'exécuter pour toujours** !

Par exemple :

```
x = 0
while x != 5:
    x += 2
```

Dans cette situation, x n'atteindra jamais 5 !

Exercice proposé :

Utilisez une boucle `for` et la fonction `range` pour calculer la somme des entiers naturels

```
9 # 1) Utilisez une boucle et la fonction "range" pour calculer la somme.
10 # Testez et récupérez le résultat en faisant tourner le code (> "Run")
11
12 for x in range (99 * 100):
13     print(x)
14
15 # 2) Assignez le résultat obtenu dans la variable "solution" pour vérification
16 solution = 4950
17
18 # Ne touchez pas le print ci-dessous :)
19 print(f"{solution} est la bonne valeur de la somme !" if solution == [(99 * 100) / 2 else "Rate"])
```

En résumé

- Les **boucles** vous permettent de **répéter des tâches facilement ou d'exécuter du code pour chaque élément dans une liste.**
- Une **boucle for** permet de **répéter du code un certain nombre de fois.**
- Une **boucle while** permet de **répéter du code jusqu'à ce qu'une certaine condition soit remplie.**

Regroupez des tâches en utilisant des fonctions

Qu'est ce qu'une fonction ?

Une **fonction** est un **bloc de code** avec un but spécifique auquel vous pouvez donner un nom. Quand vous appelez cette **fonction**, vous exécutez le **code** qu'elle contient. Les **fonctions** vous laissent **saisir des paramètres** pour **exécuter le même code sur différentes valeurs**.

Il y a différents types de **fonctions** dans Python :

- Les **fonctions intégrées** fournies avec Python.
- Les **fonctions définies** par l'utilisateur que les **développeurs** (vous !) **créent**.

Vous avez déjà utilisé une **fonction intégrée** ! `range()` est une **fonction** dans laquelle vous pouvez saisir un nombre et elle renverra ou répondra avec une séquence allant de 0 à -1.

Définissez une fonction

Pensez à une **fonction** comme à un **moyen de réutiliser un ensemble d'instructions répétables**. Vous la définissez avec le mot-clé **def**, le nom de la **fonction**, des **parenthèses** et **deux-points** : . Si la **fonction** a besoin d'un **paramètre** ou plus, il faut les saisir à l'intérieur des **parenthèses, séparés par des virgules**. Admettons que vous voulez ajouter deux nombres ensemble. L'extrait de code ci-dessous est une méthode `add()` qui prend deux nombres comme paramètres et renvoie la somme.

```
def add(a, b):
    return a + b
```

Maintenant que vous connaissez les **fonctions**, vous pouvez les **appeler** en y saisissant des **valeurs** en tant que **paramètres**. Ces **valeurs** sont appelées **arguments**. La **fonction** renverra les **valeurs ajoutées** ensemble.

```
>> add(1,3)
4
>> add(403,123)
526
```

Quand une valeur est renvoyée dans une **fonction**, vous pouvez la sauvegarder dans une **variable**.

```
total = add(1,4)
```

Dans ce cas, qu'est-ce que **total** va donner comme résultat ?

Si vous avez deviné 5, vous avez juste ! 😊 Avec la fonction `add()` vous pouvez saisir deux nombres et avoir la somme en retour.

Exercice

Faire la somme de chiffres dans un tableau :

```
fonctionAddition.py > ...
1 def addition(listes_nombres):
2     resultat = 0
3     for chiffre in listes_nombres:
4         resultat = resultat + chiffre
5     return resultat
6
7 listes_nombres = [42, 16, 24, 30, 14]
8
9 print (addition(listes_nombres))

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL CODEWHISPERER REFERENCE LOG JUPYTER

PS C:\Users\aresv\Documents\projetOpenClassroom\Python> & C:/Users/aresv/AppData/Local/Microsoft/Windows/126
PS C:\Users\aresv\Documents\projetOpenClassroom\Python> []
```

En résumé

- Les **fonctions** sont un moyen de **répéter des fonctionnalités** et de **séparer du code** dans des **modules différents**.
- Vous pouvez créer des **fonctions** avec ou sans **paramètres d'entrée**.
- Les **fonctions** sont **définies (code écrit)**, **appelées (code exécuté)** et peuvent **renvoyer des informations** (une valeur est donnée comme résultat).

Écrivez du code correctement

Résumé des bonnes pratiques du code :

- **Écrire du code standard et structuré** vous **aide**, **vous et d'autres développeurs**, à avoir une **compréhension cohérente et partagée** de ce que le **code** est censé faire.
- **DRY** : ne vous répétez pas dans le code ! **Copier et coller est interdit** !
- **Responsabilité unique** : chaque **fonction** a un **but unique** pour une **meilleure lisibilité**.

Importez des packages Python

C'est quoi les packages Pythons ?

Un **package** (une bibliothèque en français) est une **collection de fonctions** qui peuvent être **ajoutées au code Python** et **appelées au besoin**, comme **n'importe quelle autre fonction**. Il n'y a aucune raison de réécrire du code pour **réaliser une tâche standard**. Avec les **packages**, vous pouvez **importer des fonctions préexistantes et étendre efficacement les fonctionnalités de votre code**.

Voici quelques exemples de packages populaires et leurs fonctions :

- **Requests**: un package **HTTP** élégant et simple pour Python. Fréquemment utilisé pour les appels d'interface REST.
- **Beautiful Soup** (ressource anglais) : un package pour récupérer des données de fichiers **HTML** et **XML**.
- **Pandas** (ressource anglais) : un outil open source rapide, puissant et accessible pour l'**analyse** et la **manipulation de données**.

Des milliers de packages Python sont disponibles pour votre code !

Installez les packages avec Pip

Pip est un **gestionnaire de packages Python**.

Qu'est-ce qu'un **gestionnaire de packages** ?

Un **gestionnaire de packages** est un outil qui permet **d'installer** et de **gérer des packages supplémentaires** dans votre terminal. **Pip** est déjà installé dans Python, comme ça vous êtes prêt à vous y mettre !

Pour installer un package avec pip dans votre terminal, utilisez la méthode suivante :

pip install <nom-du-package>

Pour voir les packages déjà installés, vous pouvez écrire le code qui suit :

pip freeze

Il va afficher une liste de tous les packages existants, qu'on appelle dépendances, dans votre terminal.

Regardez la capture vidéo ci-dessous pour savoir comment installer et voir vos packages Python avec pip !

Importez les packages dans votre code

Pour importer une **fonctionnalité** du package dans votre code, utilisez le mot-clé **import**

En utilisant le mot-clé **import** tout en haut de votre fichier de code, vous pourrez **importer certaines fonctions d'un package ou le package entier d'un coup**. Par exemple, pour importer le **package Requests entier**, vous devez d'abord installer **Requests** avec votre **terminal** en utilisant **pip install requests**. Ensuite vous pouvez écrire le code suivant dans votre éditeur de texte :

import requests

Après, pour appeler la **fonction get()** dans le package **Requests**, écrivez :

requests.get()

Si vous voulez **importer une seule fonction** appelée **get** à partir du package **Requests**, écrivez ceci :

from requests import get

Maintenant vous n'avez plus besoin d'écrire **requests** avant **get()** parce que la **méthode** elle-même a déjà été **importée**.

En résumé :

- Un **package** est une **collection de fonctions** existantes qui peut être utilisée dans votre code.
- **Pip** est un **gestionnaire de packages Python** qui vous permet d'installer et de gérer des packages externes.
- Le mot-clé **import** permet d'**importer des packages entiers ou des fonctions** spécifiques d'un **package** dans votre code.

Extrayez et transformez des données avec l'extraction web

Qu'est-ce que l'extraction de données web ?

L'**extraction de données web** est un **processus automatique de récupération** (ou **extraction**) **des données d'un site web**. Au lieu de **collecter des données manuellement**, vous pouvez écrire des **scripts Python** (une façon contournée de dire des **processus de code**) qui peuvent **collecter des données à partir d'un site web et les sauvegarder dans des fichiers .txt ou .csv**.

Imaginons que vous soyez expert marketing. Vous êtes en train de préparer une campagne pour un nouveau type de veste. Ce serait utile de collecter des informations comme le prix et la description de vestes similaires. Au lieu de rechercher et copier/coller manuellement les informations dans un tableur, vous pouvez écrire du code Python pour collecter des données automatiquement à partir d'internet et les sauvegarder dans un fichier CSV.

Dans les deux prochains chapitres, je vais vous guider pas à pas dans un exercice d'extraction de données. Vous allez apprendre de nouvelles choses et pratiquer certains outils que vous avez déjà utilisés avant, comme les fonctions et les variables. Assurez-vous de suivre les indications de votre éditeur. Vous apprendrez mieux si vous réalisez les étapes de votre côté en même temps !

Pour cet exercice d'extraction, nous allons extraire des nouvelles et des communications du site web d'informations et des services du gouvernement du Royaume-Uni (<https://gov.uk>), transformer les données dans le format désiré et charger les données dans un fichier CSV.



ETL : Extraire, Transformer, Charger

ETL signifie **extraction, transformation et chargement** (**Extract, Transform, Load** en anglais). C'est une procédure qui "permettant d'effectuer des synchronisations massives d'information d'une source de données (le plus souvent une base de données) vers une autre" (source Wikipédia). C'est une manière compliquée de nommer le **processus** qui sert à récolter des données à un endroit, à les manipuler un peu et à les sauvegarder dans un autre endroit.

L'**extraction de données web** est une **forme d'ETL** : vous extrayez des données à partir d'un **site web**, vous les transformez dans le format que vous voulez et vous les chargez dans un fichier **CSV** (ou dans **une base de données**).

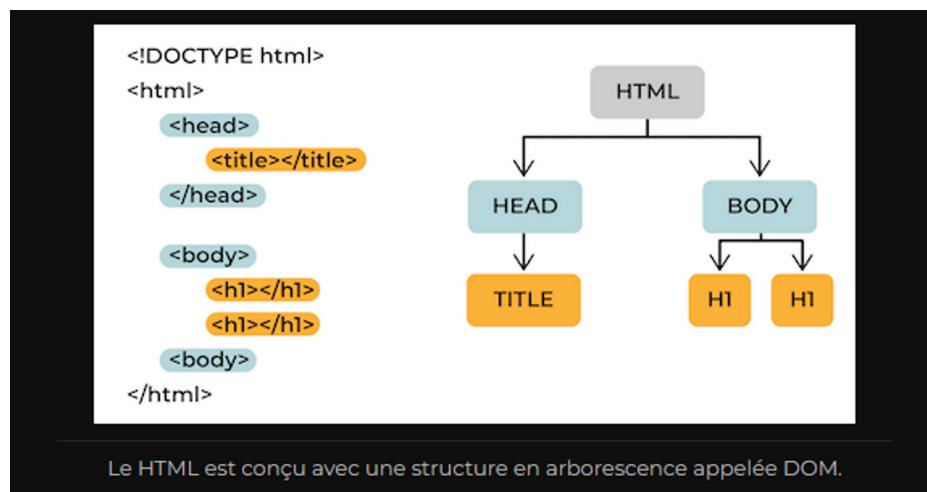
Pour extraire les données à partir d'internet, vous devez connaître quelques bases à propos du HTML, la structure de chaque page internet que vous voyez sur internet. Si vous n'avez encore jamais utilisé HTML, ne vous inquiétez pas, ce chapitre contient tout ce qu'il faut savoir pour l'extraction.

Lisez les balises HTML essentielles

Le **HTML** est le langage utilisé pour toutes les pages internet que vous voyez sur internet. Si vous faites un clic droit sur n'importe quel site web (même ici) et que vous sélectionnez Voir la page source (ou Afficher le code source), vous verrez le code **HTML** utilisé pour afficher ce que vous voyez.

Le **HTML** est conçu avec une structure en arborescence appelée **DOM (Document Object Model)**. La structure **DOM** comprend différentes balises qui peuvent s'emboîter les unes les autres. **Certaines balises représentent chaque partie d'une page HTML** et la plupart des éléments ont des balises d'ouverture et de fermeture.

Une balise d'ouverture ressemble à ça : **<nom_element>**. Une balise de fermeture a le même **nom_element**, mais avec **/** devant : **</nom_element>**. Par exemple, chaque page a une balise d'ouverture **<html>** et une balise de fermeture **</html>**. Toutes les informations que vous voulez dans cet élément doivent être entre ces deux balises.



De nombreux types de balises existent, chacun représentant différents éléments que vous pouvez ajouter à la page. Voici les plus classiques :

<p> pour un **paragraphe**.

**** pour un **élément en gras**.

<a> pour un **lien hypertexte**.

<div> une **nouvelle section ou division**.

La balise **HTML** représente le niveau supérieur de l'arborescence **DOM** et le **reste des balises** sont comme les **enfants des balises parentes correspondantes**.

Deux choses importantes à connaître sur les balises HTML : les attributs **class** et **id** qui donnent des identifiants à différents éléments **HTML**. Par exemple, si vous voulez identifier tous les éléments de « **vetements** » dans un seul identifiant, vous pouvez écrire le code ci-dessous :

```
<p class="vetements">chemise</p>
<p class="vetements">chaussettes</p>
```

De cette manière, vous savez que tous les éléments avec la classe « **vetements** » contiendront un élément lié aux vêtements à l'intérieur. Vous pouvez utiliser cette classe « **vetements** » plus tard pour que tous les éléments soient marqués avec la même classe.

De la même manière, pour avoir tous les titres et les descriptions du site d'informations et de services britanniques, nous pouvons trouver la class ou l'id de chacun de ces éléments. Nous pouvons utiliser le bouton « **Voir la page source** » pour voir le code HTML de la page et chercher l'identifiant voulu.

Voici un exemple de code HTML que nous voulons extraire de la page web :

```
1 <li class="gem-c-document-list__item">
2   <a data-e-commerce-path="/government/news/restart-of-the-uk-in-japan-campaign--2" data-e-commerce-
row="1" data-e-commerce-index="1" data-track-category="navFinderLinkClicked" data-track-action="News
and communications.1" data-track-label="/government/news/restart-of-the-uk-in-japan-campaign--2"
data-track-options='{"dimension28":20,"dimension29":"Restart of the UK in JAPAN campaign"}'
class="gem-c-document-list__item-title gem-c-document-list__item-link"
href="/government/news/restart-of-the-uk-in-japan-campaign--2">
3   | Restart of the UK in JAPAN campaign
4   </a>
5
6   <p class="gem-c-document-list__item-description">
7     | The British Embassy, British Consulate-General and the British Council, in partnership with
principal partners Jaguar Land Rover and Standard Chartered Bank are proud to announce the resumption
of our ambitious UK in JAPAN c...
8   </p>
9 </li>
```

Puisque cette information est disponible sur le web, nous pouvons écrire un script **Python** pour extraire les données que nous voulons directement depuis la page. Nous allons nous aider de **Requests** et de **Beautiful Soup** !

Le package Requests

Pour extraire des données à partir d'un site web, nous devons utiliser le package **Requests**. Rappelez-vous qu'elle fournit la fonctionnalité de faire des requêtes **HTTP**. Nous pouvons l'utiliser puisque nous essayons d'obtenir des données à partir d'un site web qui utilise le protocole **HTTP** (par exemple, <http://google.com>).

Le package **Requests** contient une fonction **.get()** qui peut être utilisée pour récupérer le code **HTML** du site.

Pour appliquer ça à l'exercice d'extraction de données web, nous allons utiliser le **package Requests** pour obtenir le code **HTML** de la page d'informations et de communication britanniques. Dans le code ci-dessous, nous **importons le package**, nous **sauvegardons l'URL** que nous voulons extraire dans une variable **url**, et nous utilisons la méthode **.get()** pour récupérer les données **HTML**. Si vous exécutez le code ci-dessous, vous verrez le code source **HTML** affiché dans la console.

```
scrappingUK.py > ...
1 import requests
2 url = "https://www.gov.uk/search/news-and-communications"
3 page = requests.get(url)
4 # Voir le code html source
5 print(page.content)
```

Même si nous avons tout le code HTML sauvegardé dans notre code, c'est encore **incompréhensible**. Il nous faut encore savoir comment **parser** les éléments exacts que nous voulons. Et nous pouvons utiliser **Beautiful Soup** pour faire ça !

Parser signifie « **parcourir le contenu d'un texte ou d'un fichier en l'analysant pour vérifier sa syntaxe ou en extraire des éléments.** »

Le package Beautiful Soup

Maintenant que nous avons le code source HTML, nous devons le **parser**. On parse le HTML avec les attributs HTML **class** et **id** mentionnés plus tôt.

Nous pouvons utiliser **Beautiful Soup** pour trouver les éléments qui peuvent être identifiés avec la class et l'**id** que nous voulons trouver. Comme pour n'importe quel package, nous allons utiliser **pip** pour installer **Beautiful Soup**.

pip install beautifulsoup4

Ensuite, nous allons importer Beautiful Soup et créer un objet « **soup** » à partir du HTML que nous avons eu avec les requêtes :

```
import requests
from bs4 import BeautifulSoup
url = "https://www.gov.uk/search/news-and-communications"
page = requests.get(url)
soup = BeautifulSoup(page.content, 'html.parser')
```

```
scrappingUK.py > ...
1 import requests
2 from bs4 import BeautifulSoup
3 url = "https://www.gov.uk/search/news-and-communications"
4 page = requests.get(url)
5 # Voir le code html source
6 print(page.content)
7 soup = BeautifulSoup(page.content, 'html.parser')
```

La variable **soup** que nous avons créée avec **Beautiful Soup** possède toutes les fonctions qui facilitent l'obtention de données à partir de HTML. Avant de récupérer les données de la page d'informations et de communications britanniques, nous allons parcourir certaines fonctionnalités de Beautiful Soup avec l'extrait HTML ci-dessous.

html

```
1 <html>
2   <head><title>Les chiens les plus mignons</title></head>
3   <body>
4     <p class="title"><b>Les meilleures races de chiens</b></p>
5
6     <p class="chiens">Il existe de nombreuses races de chiens géniales et les meilleures sont :
7       <a href="http://exemple.com/labradoodle" class="race" id="lien1">LabraDoodle</a>,
8       <a href="http://exemple.com/retriever" class="race" id="lien2">Golden Retriever</a> et
9       <a href="http://exemple.com/carlin" class="race" id="lien3">Carlin</a>
10    </p>
11
12   </body>
13 </html>
```

Une fois qu'on a créé un objet « soup » à partir du HTML, nous pouvons accéder à tous les éléments de la page très facilement !

python

```
1 # Récupération du titre de la page HTML
2 >> soup.title
3 <title>Les chiens les plus mignons</title>
4
5 # Récupération de la chaîne de caractères du titre HTML
6 >> soup.title.string
7 "Les chiens les plus mignons"
8
9 # Trouver tous les éléments avec la balise <a>
10 >> soup.find_all('a')
11 [ <a href="http://exemple.com/labradoodle" class="race" id="lien1">LabraDoodle</a>,
12   <a href="http://exemple.com/retriever" class="race" id="lien2">Golden Retriever</a>,
13   <a href="http://exemple.com/carlin" class="race" id="lien3">Carlin</a>]
14
15 # Trouver les éléments avec l'id du « lien1 »
16 >> soup.find(id="lien1")
17 <a href="http://exemple.com/labradoodle" class="race" id="lien1">LabraDoodle</a>
18
19 # Trouver tous les éléments p avec la classe « title »
20 >> soup.find_all("p", class_="title")
21 "Les meilleures races de chiens"
```

Et ce n'est qu'un aperçu de la façon dont Beautiful Soup peut vous aider à obtenir facilement les éléments spécifiques que vous voulez à partir d'une page HTML. Vous pouvez obtenir des éléments par leur balise, ID ou classe.

Et ce n'est qu'un aperçu de la façon dont **Beautiful Soup** peut vous aider à obtenir facilement les éléments spécifiques que vous voulez à partir d'une page **HTML**. Vous pouvez obtenir des éléments par leur **balise**, **ID** ou **classe**.

Appliquons cela à l'exercice du site d'informations et de services britanniques. Nous avons déjà converti la page en objet « **soup** » avec cette ligne **soup = BeautifulSoup(page.content, 'html.parser')** .

Voyons maintenant quelles données nous pouvons obtenir de la page d'informations et communications. D'abord, allons récupérer les titres de toutes les histoires. Après avoir inspecté la page **HTML**, nous pouvons voir que tous les titres sont dans les éléments « **lien** » indiqués par la **balise <a>** et ont la même classe : **gem-c-document-list__item-title**.

Voici un exemple :

```

1 <a data-e-commerce-path="/government/news/restart-of-the-uk-in-japan-campaign--2" data-e-commerce-
   row="1" data-e-commerce-index="1" data-track-category="navFinderLinkClicked" data-track-action="News
   and communications.1" data-track-label="/government/news/restart-of-the-uk-in-japan-campaign--2"
   data-track-options='{"dimension28":20,"dimension29":"Restart of the UK in JAPAN campaign"}'
   class="gem-c-document-list__item-title gem-c-document-list__item-link"
   href="/government/news/restart-of-the-uk-in-japan-campaign--2">Restart of the UK in JAPAN
   campaign</a>

```

Nous pouvons utiliser les deux ensemble pour avoir une liste de tous les éléments **title** :

```
titres = soup.find_all("a", class_="gem-c-document-list__item-title")
```

On obtient une liste de tous les éléments avec la classe **gem-c-document-list__item-title**. Pour voir la valeur de la chaîne dans un élément, nous pouvons parcourir chaque élément dans la liste et afficher la chaîne de l'élément.

python

```

1 >> for title in titres:
2 ...     print(title.string)
3
4 "Restart of the UK in JAPAN campaign"
5 "Joint Statement on the use of violence and repression in Belarus"
6 "Foreign Secretary commits to more effective and accountable aid spending under new Foreign,
  Commonwealth and Development Office"
7 "UK military dog to receive PDSA Dickin Medal after tackling Al Qaeda insurgents."

```

Les descriptions sont dans une balise **<p>** et ont la classe **gem-c-document-list__item-description**, comme vous pouvez le voir ici :

HTML

```

1 <p class="gem-c-document-list__item-description">Joint Statement by the Missions of the United
  States, the United Kingdom, Switzerland and the European Union on behalf of the EU Member States
  represented in Minsk on the use of violence and repression in Belarus</p>

```

Pour avoir toutes ces descriptions, écrivez :

python

```
1 descriptions = soup.find_all("p", class_="gem-c-document-list__item-description")
```

```

❸ scrappingUK.py > ...
6 #print(page.content)
7 soup = BeautifulSoup(page.content, 'html.parser')
8 titres = soup.find_all("a", class_="gem-c-document-list__item-title")
9 descriptions = soup.find_all("p", class_="gem-c-document-list__item-description")
10
11 #Affichage des titres
12
13 for title in titres :
14     print (title.string)
15
16 #Affichage des descriptions
17
18 for description in descriptions:
19     print (description.string)

```

Vous transformez des données quand vous les convertissez d'un format à l'autre. Ça peut être aussi simple que de convertir une chaîne de caractères en une liste, mais ça peut aussi concerter la transformation de milliers de listes en dictionnaires. Généralement, ça nécessite d'associer différents points de données. Il y a plusieurs façons de transformer des données et la décision dépend finalement du type de données et du format que vous voulez avoir.

Voici quelques exemples de transformation de données :

- Convertir un format de champ de date de « 28 décembre 2019 » à 28/12/19.
- Convertir une somme d'argent de dollars en euros.
- Standardiser les adresses e-mail ou postales.

Pour l'exemple des **informations et communications britanniques**, nous allons sauvegarder tous les titres et les descriptions de la page HTML dans une liste de chaînes de caractères. Au lieu d'afficher les informations des chaînes dans le terminal comme vous venez de le faire, nous voulons sauvegarder les éléments dans une liste :

Ici nous avons commencé avec une **liste vide** appelée **titres**. Ensuite nous avons fait une **boucle** pour tous les éléments dans la liste **titres_bs** et ajouté uniquement la chaîne des titres dans la liste « **titres** ». Maintenant, la liste **titres** contiendra une liste de chaînes de tous les titres de la page HTML.

Suivez la même approche pour extraire et sauvegarder les descriptions de la page.

python

```
1 descriptions_bs = soup.find_all("p", class_="gem-c-document-list__item-description")
2 descriptions = []
3 for desc in descriptions_bs:
4     descriptions.append(desc.string)
```

En résumé :

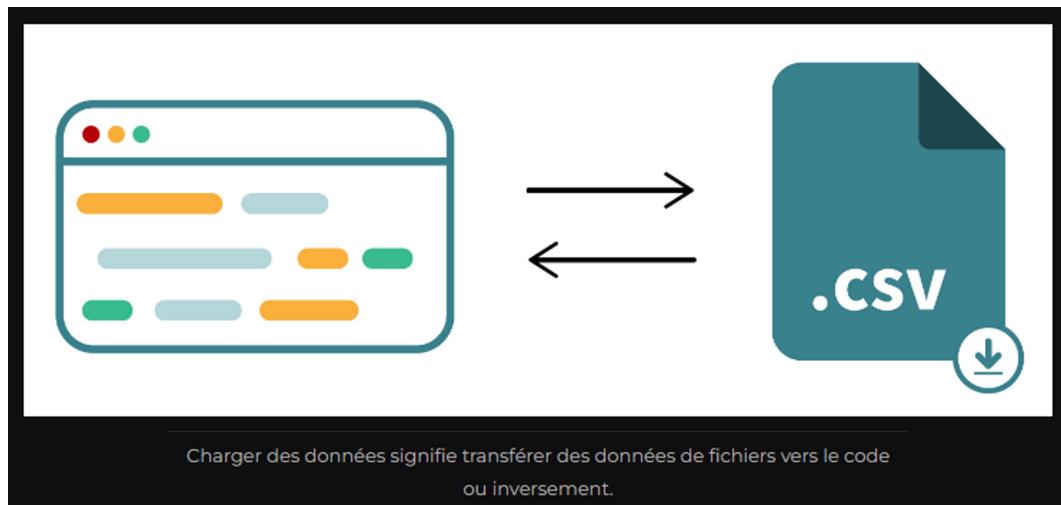
- L'**extraction de données web** est un processus automatisé de récupération des données d'internet.
- **ETL** signifie **extraction, transformation et chargement**. C'est un acronyme très utilisé dans la programmation, pour désigner le processus de récupération de données d'un endroit, de **modification** légère de ces données et de leur **sauvegarde dans un autre endroit**.
- **HTML** est la **structure de n'importe quelle page web** et la **compréhension de cette structure** va vous aider à savoir comment récupérer les données dont vous avez besoin.
- **Requests** et **Beautiful Soup** sont des **packages Python** tiers qui peuvent vous aider à **récupérer et parser** les données d'internet.
- **Parser des données** signifie les préparer pour les transformer, les sauvegarder ou les utiliser.

Chargez des données avec Python

Charger des données. Ça veut dire quoi ?

Python peut être utilisé pour lire les données de différents endroits, y compris les bases de données et les fichiers. Deux types de fichiers sont souvent utilisés : .txt et .csv. Vous pouvez importer et exporter les fichiers avec la

fonctionnalité intégrée de Python ou le package CSV. Nous allons voir les deux options !



Chargez des données avec les fonctions intégrées de Python

Pour lire et écrire un fichier, vous pouvez utiliser la fonction intégrée `open()`, qui requiert deux paramètres : le nom du fichier et le mode.

- Nom du fichier : le chemin d'accès au fichier que vous voulez lire ou dans lequel vous voulez écrire.
- Mode : le mode que vous voulez utiliser pour le fichier. Les options principales sont :
 - Lire : "`r`"
 - Écrire (écraser) : "`w`"
 - Continuer d'écrire : "`a`"
 - Lire et écrire (écraser) : "`r+`"

Pour créer un nouveau fichier appelé « bonjour.txt » et y écrire « Hello, world! », utilisez le code ci-dessous :

```
fichier = open("hello.txt", "w")
fichier.write("Hello, world!")
fichier.close()
```

Vous pouvez aussi utiliser l'instruction `with` pour fermer automatiquement le fichier à la fin du bloc :

```
with open("file.txt") as fichier:
    for ligne in fichier:
        # faire quelque chose avec une ligne
        print(ligne)
```

Avec ce code, le fichier d'entrée va être affiché ligne par ligne. Vous avez probablement remarqué que nous n'avons pas spécifié de mode dans `open()` ... C'est tout simplement parce que le mode d'ouverture par défaut est la lecture ou "`r`" !

Le package CSV

La méthode **open()** peut lire et écrire sur les fichiers **.txt** et **.csv**, mais vous pouvez aussi utiliser le **package CSV de Python** pour lire et écrire dans les **fichiers CSV**. Ce package offre plus de fonctionnalités.

La documentation officielle de Python sur le format CSV explique que le package CSV « vous permet de dire « écris ces données dans le format préféré par Excel » ou « lis les données de ce fichier généré par Excel », sans connaître les détails précis du format CSV utilisé par Excel. »

Quand vous utilisez le **package CSV**, vous devez aussi utiliser la fonction **open()** pour ouvrir le fichier. Vous pouvez ensuite utiliser les méthodes **reader()** ou **writer()** sur le fichier pour le **lire** ou y **écrire**.

Lisez les fichiers externes

Commençons avec la **lecture** des fichiers externes. Disons que vous avez un fichier **CSV** nommé **couleurs_preferees.csv** qui ressemble à ça :

```
nom,metier,couleur_preferee  
Jacob Smith,Ingénieur en informatique,Violet  
Nora Scheffer,Stratégiste numérique,Bleu  
Emily Adams,Responsable marketing,Orange
```

La méthode **.reader()** va prendre tout le texte dans un CSV, le **parser ligne par ligne** et **convertir chaque ligne dans une liste de chaînes**. Vous pouvez utiliser **différents délimiteurs** pour décider de la manière de **séparer chaque colonne**, mais le **séparateur le plus commun est une virgule**. L'extrait de code ci-dessous lit le fichier **CSV** et affiche chaque ligne.

```
import csv  
  
with open('couleurs_preferees.csv') as fichier_csv:  
    reader = csv.reader(fichier_csv, delimiter=',')  
    for ligne in reader:  
        print(ligne)
```

Le résultat sera comme ceci :

```
['nom', 'metier', 'couleur_preferee']  
['Jacob Smith', 'Ingénieur en informatique', 'Violet']  
['Nora Scheffer', 'Stratégiste numérique', 'Bleu']  
['Emily Adams', 'Responsable marketing', 'Orange']
```

Bien que cette approche peut être utile parfois, la ligne d'en-tête est considérée comme la même que les autres. Une méthode plus utile pour lire les **fichiers CSV**, tout en reconnaissant les en-têtes pour identifier les colonnes, est la **méthode DictReader()**. Cette méthode sait que la première ligne est un en-tête et sauvegarde les autres lignes en tant que dictionnaires. Chaque clé est un nom de colonne et la valeur est la valeur de la colonne.

Le code ci-dessous montre comment utiliser la méthode **DictReader()**.

```
import csv  
  
with open('couleurs_preferees.csv') as fichier_csv:  
    reader = csv.DictReader(fichier_csv, delimiter=',')  
    for ligne in reader:
```

```
print(ligne['nom'] + " travaille en tant que " + ligne['metier'] + " et sa couleur préférée est " +  
ligne['couleur_preferee'])
```

Le résultat affichera :

Jacob Smith travaille en tant que Ingénieur en informatique et sa couleur préférée est Violet
Nora Scheffer travaille en tant que Stratégiste numérique et sa couleur préférée est Bleu
Emily Adams travaille en tant que Responsable marketing et sa couleur préférée est Orange

Écrivez dans des fichiers externes

Pour comprendre comment écrire dans des **fichiers externes**, revenons sur notre exemple d'**extraction de données web**. Nous avons déjà écrit le **code** pour **extraire et transformer les données** du site d'informations et services du gouvernement britannique. Nous avons sauvegardé tous les titres et descriptions dans des listes de chaînes de caractères. Maintenant nous pouvons utiliser les fonctions **.writer()** et **.writerow()** pour **écrire les données** dans le **fichier CSV**.

```
# Créer une liste pour les en-têtes  
en_tete = ["titre", "description"]  
  
# Créer un nouveau fichier pour écrire dans le fichier appelé « data.csv »  
with open('data.csv', 'w') as fichier_csv:  
    # Créer un objet writer (écriture) avec ce fichier  
    writer = csv.writer(fichier_csv, delimiter=',')  
    writer.writerow(en_tete)  
    # Parcourir les titres et descriptions - zip permet d'itérer sur deux listes ou plus à la fois  
    for titre, description in zip(titres, descriptions):  
        # Créer une nouvelle ligne avec le titre et la description à ce moment de la boucle  
        ligne = [titre, description]  
        writer.writerow(ligne)
```

Ensuite téléchargez le fichier de code **script_p3c3.py** de [ce dossier](#) et exécutez-le dans votre éditeur. Prenez le temps de comprendre ce que chaque ligne fait et n'hésitez pas à regarder les captures vidéos plusieurs fois si besoin.

Vous avez peut-être remarqué que certaines instructions de ce code se répètent. Essayez de les séparer en différentes **fonctions** vous-même. Une fois que vous avez essayé, regardez le fichier de code **script_p3c3_solution.py** du [dossier](#) pour comparer avec ce que vous avez fait. Mais n'oubliez pas qu'il n'y a pas qu'une seule bonne réponse !

En résumé :

- On charge des données en lisant un fichier ou en y écrivant.
- On peut lire et écrire des **fichiers** avec la **méthode intégrée** de **Pythonopen()**
- Les méthodes **.writer()** et **.DictReader()** du **package CSV** de Python facilitent le travail sur les **fichiers CSV**.
- Les modes principaux d'écriture de fichiers sont « **r** » pour la lecture, « **w** » pour l'écriture (écrasement) et « **a** » pour l'ajout.

Quels sites web sont extractibles ?

Les sites web n'ont pas tous des politiques qui autorisent d'extraire leurs données. Pour savoir si une entreprise

l'autorise, assurez-vous de vérifier leurs Conditions générales.

Pour de nombreux sites, ceux-ci sont disponible sur leur fichier robots.txt. Vous pouvez ajouter « /robots.txt » après l'URL du site pour voir s'ils en ont un.

Ici vous pouvez voir une capture d'écran de la page robots.txt de Facebook, qui indique très clairement (en anglais) que l'extraction est interdite :

```
# Notice: Collection of data on Facebook through automated means is
# prohibited unless you have express written permission from Facebook
# and may only be conducted for the limited purpose contained in said
# permission.
# See: http://www.facebook.com/apps/site_scraping_tos_terms.php

User-agent: Applebot
Disallow: /ajax/
Disallow: /album.php
Disallow: /checkpoint/
Disallow: /contact_in [Click to go back, hold to see history]
Disallow: /dialog/
Disallow: /fbml/ajax/dialog/
Disallow: /feeds/
Disallow: /file_download.php
Disallow: /hashtag/
Disallow: /l.php
Disallow: /moments_app/
Disallow: /p.php
Disallow: /photo.php
Disallow: /photos.php
Disallow: /share.php
Disallow: /share/
Disallow: /sharer.php
Disallow: /sharer/
```

La collecte de données sur Facebook est interdite par le site web.

Question 1

Quelle est la syntaxe correcte pour installer le package `requests` avec pip ?

Option 1

```
python  
1 install requests
```

Option 2

```
python  
1 pip install package requests
```

✓ Option 3

```
python  
1 pip install requests
```

Option 4

```
python  
1 pip import requests
```

Option 5

```
python  
1 import requests
```

Pour installer un package, `pip` doit être en début de ligne, suivi de `install` et du nom du package.

Question 2

Quelle est la meilleure définition de l'extraction de données web, parmi les propositions suivantes ?

- Le fait de récupérer des données de différentes bases de données et de les enregistrer dans un fichier .csv ou .txt.
- Le fait d'extraire des données de nombreuses sources différentes, de les transformer, puis de les charger dans une base de données.
- ✓ Un processus automatisé de récupération de données depuis le web.
- Un processus qui permet aux individus de ne pas avoir à copier-coller manuellement des données d'un site web.
- Un processus automatisé pour récolter des données depuis plusieurs ordinateurs, et les sauvegarder.

L'extraction de données web est un processus automatisé d'extraction et de sauvegarde de données spécifiquement depuis le web.

Question 3

Les questions 3 à 6 s'appuient sur l'extrait de code HTML suivant.

```
html
1 <html>
2 <head> <title> Repas Rapides & Sains </title> </head>
3 <body>
4 <p class = "introduction"><i> Ces recettes sont parfaites lorsque vous avez peu de temps et que
vous avez besoin de manger sain. </i></p>
5 <p class= "methodes"> Lorsque vous avez peu de temps, mais que vous avez quand même envie d'un
repas sain, nourrissant et rapide, essayez ces recettes :
6 <a href="http://exemple.com/petit-dejeuner" class="recette" id="lien1">Recettes pour le petit-
déjeuner</a>,
7 <a href="http://exemple.com/dejeuner" class="recette" id="lien2">Recettes pour le déjeuner</a> et
8 <a href="http://exemple.com/diner" class="recette" id="lien3">Recettes pour le dîner</a>;
9 </p>
10 </body>
11 </html>
```

Chargez cet extrait HTML dans votre code Python avec BeautifulSoup, avec la variable nommée `soup`.

Quel est le résultat du code suivant ?

```
python
1 soup.title.string
```

- Repas Rapides & Sains
- <title>Repas Rapides & Sains</title>
- Erreur de syntaxe
- Repas
- "Repas"

Cet extrait ne renvoie que la valeur de la chaîne de caractères contenue entre les balises HTML `<title>`.

Question 4

Quel est le résultat du code suivant ?

```
python
1 soup.find_all("p", class_="introduction")
```

- "Repas Rapides & Sains"
- Repas Rapides & Sains
- Erreur de syntaxe
- Repas
- "Repas"

Vous devriez obtenir une erreur de syntaxe, car il manque un signe égal `=` entre `class_` et `"introduction"`.

Question 5

Quel est le résultat de l'extrait de code suivant ?

python

```
1 soup.find_all("a")
```

- [<a href="<http://exemple.com/petit-dejeuner>" class="recette" id="lien1">Recettes pour le petit-déjeuner, <a href="<http://exemple.com/dejeuner>" class="recette" id="lien2">Recettes pour le déjeuner, <a href="<http://exemple.com/diner>" class="recette" id="lien3">Recettes pour le dîner]
- [<a href="<http://exemple.com/petit-dejeuner>" class="recette" id="lien1">Recettes pour le petit-déjeuner]
- ["<http://exemple.com/petit-dejeuner>", "<http://exemple.com/dejeuner>", "<http://exemple.com/diner>"]
- ["Recettes pour le petit-déjeuner, Recettes pour le déjeuner, Recettes pour le dîner"]

`find_all("a")` renvoie une liste de tous les éléments en lien, signalés par `<a>`. Il renvoie l'élément `<a>` en entier, et non pas uniquement l'URL ou le texte.

Question 6

Qu'est-ce qui est renvoyé par `soup.find(id="lien2")` ?

- Recettes pour le déjeuner
- <a href="<http://exemple.com/dejeuner>" class="recette" id="lien2">Recettes pour le déjeuner
- <a href="<http://exemple.com/diner>" class="recette" >Recettes pour le dîner
- <a href="<http://exemple.com/petit-dejeuner>" class="recette" >Recettes pour le petit-déjeuner
- <a href="<http://exemple.com/petit-dejeuner>" class="recette" id="lien2">Recettes pour le petit-déjeuner

`soup.find(id="lien2")` renvoie l'élément entier associé à l'identifiant "lien2". Dans ce cas, c'est un élément `<a>`.

Question 7

Que pouvez-vous ajouter à la fin d'une URL, parmi les options suivantes, pour trouver la politique d'extraction de données web d'un site web ?

- '/robots.txt'
- '/policies.txt'
- '/robots.com'
- '/robots.csv'
- '/policies.com'

L'ajout de /robots.txt à la plupart des gros sites web vous permet de voir leur politique d'extraction de données web.

Question 8

Imaginons que vous ayez un document texte nommé «salut.txt» qui ressemble à ce qui suit :

```
1 Le chien est mon animal préféré.
```

python

Quelle est la syntaxe correcte pour ajouter «Mon chien s'appelle Lucky» à «salut.txt»?

Option 1

python

```
1 fichier = open("salut.txt", "a")
2 fichier.write("Mon chien s'appelle Lucky.")
```

Option 2

python

```
1 fichier = open("salut.txt", "w")
2 fichier.write("Mon chien s'appelle Lucky.")
```

Option 3

python

```
1 fichier.write("Mon chien s'appelle Lucky.")
```

Option 4

python

```
1 fichier.append("Mon chien s'appelle Lucky.")
```

Option 5

python

```
1 fichier = open("salut.txt", "a")
2 fichier.append("Mon chien s'appelle Lucky.")
```

Lorsque vous faites des ajouts, vous devez inclure `"a"` et écrire la ligne en tant que chaîne de caractères.

Option 1

```
python
1 jus = soup.find_all(id="jus")
2 soup.extraire_donnees(print(jus))
```

Option 2

```
python
1 jus = soup.find_all("id="jus")
2 print(jus.extraire_donnees())
```

Option 3

```
python
1 jus = soup.find_all(class_="jus1")
2 extraire_donnees(jus.strings)
```

Option 4

```
python
1 jus = soup.find("ids=jus")
2 extraire_donnees(soup)
```

Option 5

```
python
1 jus = soup.find_all(class_="jus")
2 print(extraire_donnees(jus))
```

Vous pouvez voir dans le HTML que tous les noms de jus ont une classe `jus`. La syntaxe pour récupérer les éléments HTML contenant le nom des jus est `soup.find_all(class_="jus")`. Puis pour obtenir une liste des noms des jus, on écrit `extraire_donnees(jus)`. Pour afficher cette liste, vous pouvez placer cet appel de fonction dans une fonction `print()`