Introduction à Python

Un atelier présenté par Clara Turp

Cet atelier utilise la pédagogie et le contenu de Library Carpentry:
https://librarycarpentry.org/lc-python-intro/

Avant l'atelier: téléchargement d'Anaconda

Installation de Python avec Anaconda

https://librarycarpentry.org/lc-python-intro/setup.html

Anaconda est un outil qui regroupe plusieurs autres outils qui vous aideront à utiliser Python.

Veuillez installer une version de Python supérieure à 3.0 (par exemple, 3.6)

Windows - Video (en anglais)

1. Ouvrez anaconda.com/download dans un navigateur web.
2. Téléchargez Python 3 pour Windows.
3. Cliquez deux fois sur le document 'exe' dans vos téléchargements et installez Python 3 en suivant presque toutes les options par défaut sauf celle-ci : sélectionnez l'option qui permet de faire d'Anaconda l'outil Python par défaut (**Make Anaconda the default Python**).

macOS - Vidéo (en anglais)

1. Ouvrez anaconda.com/download dans un navigateur web.
2. Téléchargez Python 3 pour macOS.
3. Nous recommandons de choisir l'option 'Graphical Installer'.
4. Installez Python 3 avec toutes les options par défaut.

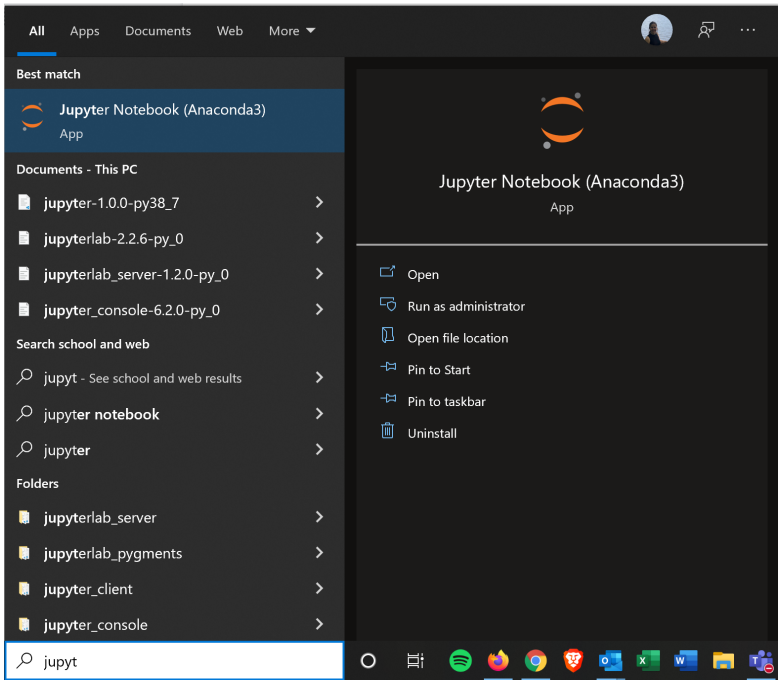
Introduction à Python

https://librarycarpentry.org/lc-python-intro/01-getting-started/index.html

Utiliser Jupyter

- Anaconda est un outil de gestion des paquets, ce qui inclut Jupyter Notebooks.

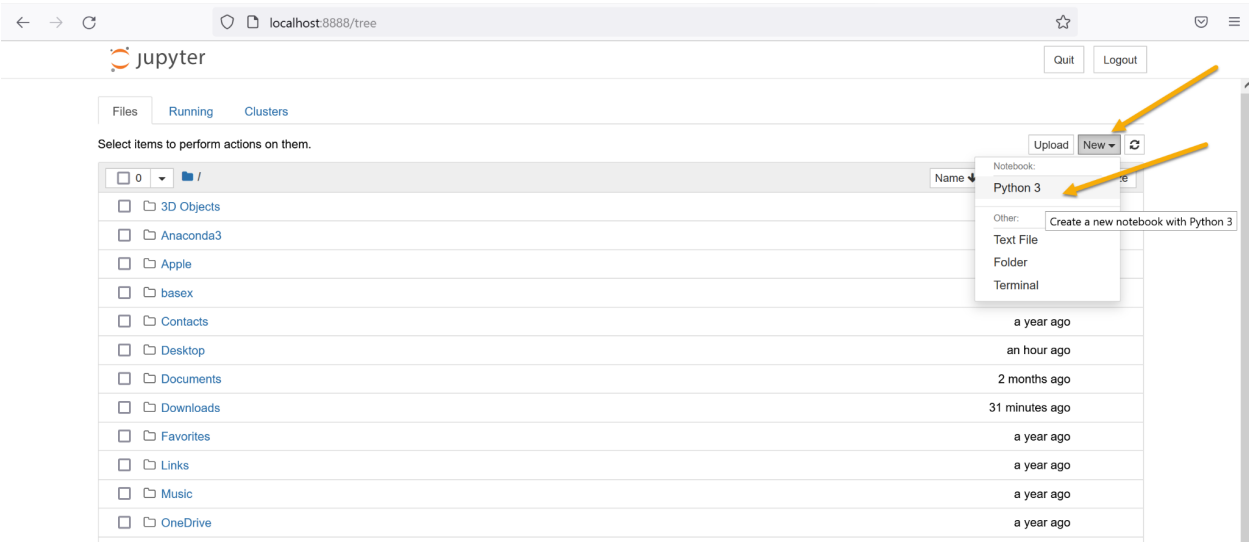
Lorsque vous aurez installé Python, cherchez dans votre ordinateur l'application Jupyter.



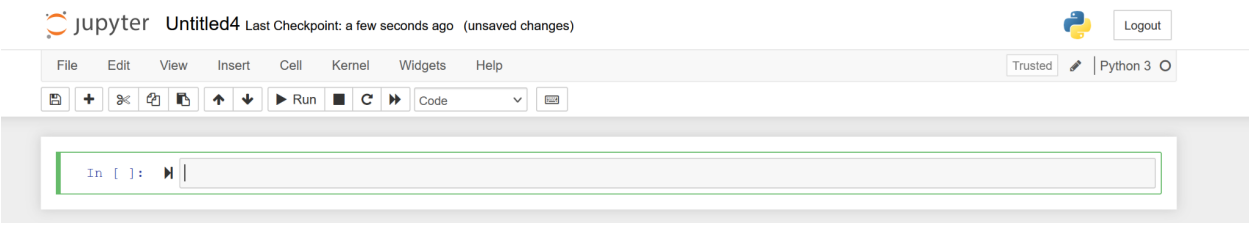
- En cliquant sur cette application, Jupyter Notebooks ouvrira dans votre navigateur web. Vous devriez aussi voir l'application exécutée dans un terminal.

Le principal avantage de Jupyter, c'est que vous pouvez créer un carnet avec du code que vous pouvez exécuter directement dans le même carnet et voir le résultat. Vous pouvez aussi ajouter des notes en utilisant le langage Markdown.

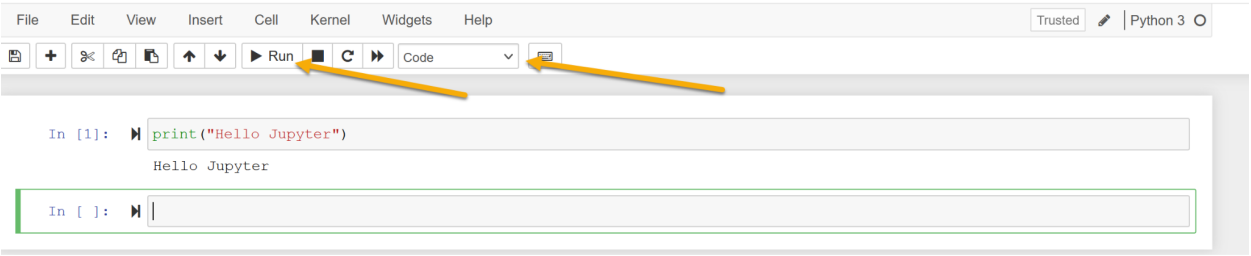
Dans Jupyter, cliquez sur **New** puis sur **Python 3 Notebook**



Un nouveau carnet sera créé dans un nouvel onglet.

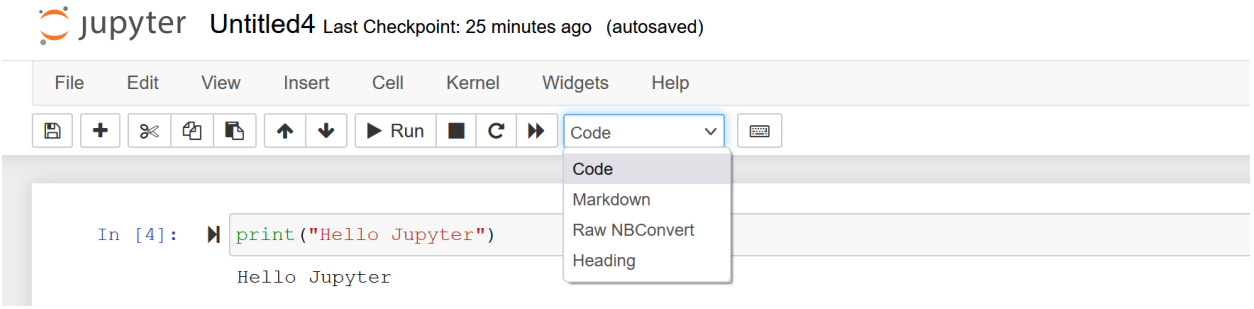


Essayez de taper du code Python dans la boîte du haut, juste après la flèche - par exemple, **print(“Hello Jupyter”)**. Vous pouvez ensuite exécuter le code en cliquant sur **Run** ou avec le raccourci clavier **Shift-Entrée**. Vous verrez ensuite le code que vous avez écrit et le résultat. Une nouvelle cellule s’ouvrira dessous. Si vous avez fait une erreur, vous pourrez éditer la cellule du dessus.



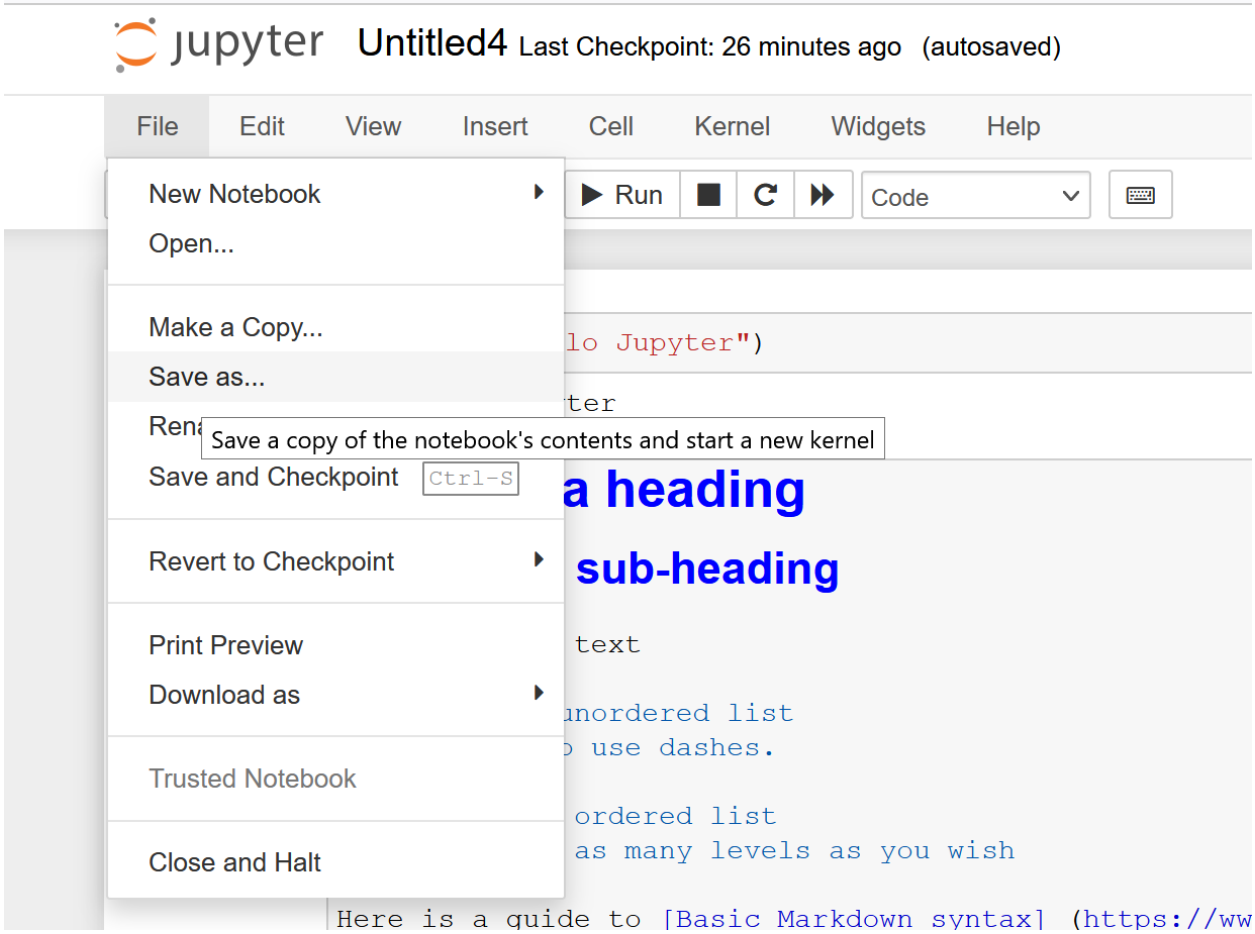
Vous pouvez aussi ajouter de nouvelles cellules en cliquant sur +, qui sert à insérer une cellule au-dessous des précédentes. Vous pouvez également sélectionner Insert et choisir si vous voulez insérer une cellule avant ou après la cellule active.

Un aspect intéressant de Jupyter Notebooks est qu'il combine différents types de cellules; vous pouvez ajouter des cellules de **code** ou des cellules **Markdown**. Cela veut dire que vous pouvez donc inclure toutes vos notes de l'atelier directement dans le code.

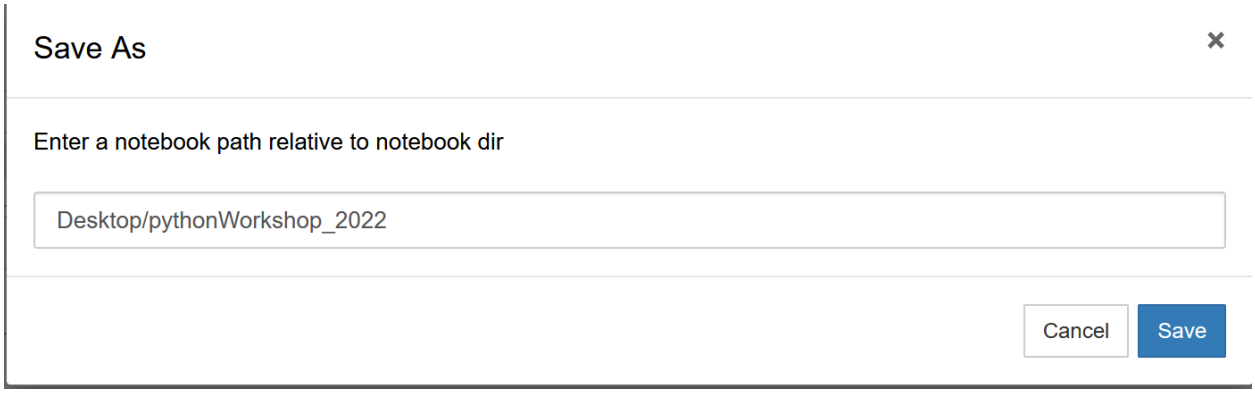


Pour sauvegarder le code

Pour sauvegarder le code, cliquez sur "file", puis choisissez "save as". Cela sauvegardera votre carnet avec l'extension ipynb. Cela sauvegardera également ce carnet en tant que répertoire actuel dans votre page d'accueil sur Jupyter.



Si je voulais sauvegarder le carnet sur mon **bureau**, il faudrait que je tape Desktop/PythonWorkshop_2022. C'est ce qu'on appelle un chemin d'accès relatif.



Utilisez des commentaires pour ajouter de la documentation à votre programme

Vous pouvez ajouter des commentaires à votre script - cela fonctionne dans Jupyter et, en fait, cela fonctionne toujours quand vous programmez en Python. Normalement, les commentaires servent à décrire ce que fait le script. C'est une forme de documentation qui pourrait aider la future version de vous-mêmes!

- Pour des commentaires d'une ligne, utilisez la touche du carré (#). La ligne de commentaire sera ignorée quand le script sera exécuté.
- Pour des commentaires de plus d'une ligne, utilisez trois guillemets anglais (""")
- Utilisez les commentaires pour déboguer.

```
#This is a comment

"""
This is a long comment

Usually when you have a multi-line comment

"""
```

Variables et affectation

https://librarycarpentry.org/lc-python-intro/02-variables/index.html

Utilisez les variables pour enregistrer des valeurs et certaines types de données

- Les variables sont des noms pour les valeurs.
- En Python, le symbole = assigne une valeur à la droite du symbole au nom situé à gauche du symbole (nom = valeur)
- La variable est créée lorsqu'une valeur lui est assignée.

Types de données :

- Nombres entiers relatifs - n'utilisez pas de guillemets.
- Chaînes de caractères (mots) - utilisez les guillemets.
- Donc, si on met un nombre entre guillemets, comme dans Titre = "1984" - de quel type est "1984"?

- Dans l'exemple ci-dessous, Python attribue un âge à une variable de type âge et un nom entre guillemets à une variable de type first_name.
- Vous voyez les différentes couleurs associées aux différents types de données?

In [ ]:

```
age = 35              #Integers (numbers)
first_name = "Clara" #Strings (words), use quotation marks.
title = "1984"        #What would this be?
```

- Si nous donnons un nom à une variable, ce nom devrait représenter la valeur désignée. Par exemple, print(age) affichera la valeur désignée par la variable âge, à savoir 35. Pour donner un nom à une variable, vous tapez ce nom sans guillemets.

```
In [1]:
age = 35              #Integers (numbers)
first_name = "Clara" #Strings (words), use quotation marks.
title = "1984"        #What would this be?

print(age)
```

```
35
```

- Les noms des variables :
    - Devraient avoir un sens clair
    - Ne peuvent pas commencer par un chiffre
    - Ne peuvent pas contenir d'espaces ou de ponctuation
    - Peuvent contenir des traits soulignés ou une casse mixte (pour séparer les mots dans un nom de variable très long)
        - Par contre, souligner au début d'une chaîne de caractères, comme dans __alistairs_real_age ont une signification particulière, alors, il faut les éviter.

Python respecte la casse.

- Python reconnaît que les majuscules et les minuscules sont différents; dès lors, il interprétera Name et name comme deux variables différentes.
- Il existe des conventions pour l'utilisation de majuscules au début des noms de variables; dès lors, nous utiliserons les minuscules pour l'instant.

Utilisez Print pour afficher des valeurs.

- Comme nous l'avons vu, Python a une fonction de base dite Print qui affiche des données sous forme de texte.
- Appelez la fonction (c'est-à-dire, dites à Python de l'exécuter) en utilisant son nom.
- Indiquez les valeurs de la fonction (c'est-à-dire, ce qui doit être affiché) entre parenthèses.
- Pour ajouter une chaîne à ce qui sera affiché, mettez la chaîne entre parenthèses.
- Les valeurs passées vers la fonction s'appellent des arguments. Donc:
    - Print est une fonction (nous y reviendrons plus tard).
    - Print requiert au moins un argument.
    - Les arguments peuvent prendre la forme de variables, de nombres, de chaînes de caractères.
    - Les arguments seront affichées sur la console.

Si nous voulons afficher Clara a 35 ans, nous pourrions taper : print("Clara is 35 years old")

Ou bien nous pourrions utiliser les variables que nous avons définies plus tôt. Veuillez noter par ailleurs que Python se souvient des variables, même si elles ont été créées plus tôt dans le processus.

Les variables doivent être créées avant d'être utilisées.

```
In [2]: # Print is a built-in function that prints the function's argument as text.
        print("Clara is 35 years old")
        print(first_name, "is", age, "years old")

        Clara is 35 years old
        Clara is 35 years old
```

Voici ce qui arrivera une fois que le script est exécuté :

- Print place automatiquement une espace entre les items pour les séparer; dans le script, les items (arguments) sont séparés par des virgules.
- Print renvoie à la ligne à la fin.

Les variables peuvent être utilisées dans des calculs.

- Nous pouvons utiliser des variables dans des calculs, tout comme s'il s'agissait de valeurs.
    - D'ailleurs, souvenez-vous que nous avons assigné la valeur 35 à la variable âge.
- Il y a ensuite plusieurs manières d'ajouter 3 à l'âge de Clara.
    - La première (age + 3) affichera 38; cependant, cette valeur ne sera pas sauvegardée.
    - La seconde, (age = age + 3) remplacera la valeur de l'âge (35) par 38.
    - La troisième, (age_plus_three = age + 3) assigne la valeur 38 à une nouvelle variable. Comme nous avons fait ces trois options, nous obtenons 41, puisque nous avons changé la valeur de la variable âge à la seconde étape.

Ci-dessous, les trois versions de l'ajout de 3 à la variable âge.

In [2]:
```
age = 35

print(age + 3)
```

```
38
```

In [3]:
```
age = 35
age = age + 3

print(age)
```

```
38
```

In [5]:
```
age = 35
age_plus_three = age + 3

print(age_plus_three, age)
```

```
38 35
```

–30–

Quelques méthodes et fonctions intégrées

https://librarycarpentry.org/lc-python-intro/04-built-in/index.html
https://www.w3schools.com/python/python_ref_string.asp

Fonctions intégrées

Les fonctions sont centrales en programmation. Vous pouvez écrire vos propres fonctions ou utiliser certaines fonctions intégrées qui sont disponibles. Les fonctions vous permettent de faire une action (ou un algorithme) avec un élément que vous lié à la fonction. Elles sont très importantes parce que vous pouvez les utiliser à répétition.

Une fonction peut nécessiter des arguments (ou pas).

- Vous avez déjà vu une fonction, mais regardons la de plus prêt.
- L'argument est la valeur qu'on associe à la fonction. Certaines fonctions n'ont pas besoin d'argument.
- Tous les arguments doivent être placés dans les parenthèses.
    - print("Je suis un argument et je dois être placé ici")
- Il faut toujours utiliser les parenthèses, c'est comme ça que Python sait que ligne représente une fonction.
    - Vous laissez les parenthèses vides si la fonction ne requiert pas d'arguments.
- Print ne requiert aucun argument, mais il est aussi possible d'en assigner plusieurs.

Les fonctions redonnent toujours quelque chose.

- Toutes les fonctions rendent un résultat.
- Si la fonction n'a rien d'utile à redonner, elle rend généralement la valeur 'None'

Utilisez la fonction len pour trouver la longueur d'un objet

- Len() est une fonction intégrée. Elle rend la longueur d'un objet.
    - Combien de caractères dans une chaîne de caractères?
    - Combien d'éléments dans une liste.
    - Cette fonction n'accepte pas les chiffres.

- Commençons par regarder la longueur d'une chaîne de caractères.
- La fonction exige un argument entre les parenthèses.

```
element = "Helium"
print(len(element))
```

```
6
```

- La priorisation des fonctions imbriquées est la même que pour les mathématiques. On évalue les fonctions de l'intérieur vers l'extérieur.
    - On commence par trouver la longueur de la variable element avant d'afficher le résultat.

Utilisez les méthodes upper() et lower() pour transformer une chaîne de caractères

- Une méthode est similaire à une fonction, mais elles sont liées à un objet particulier
    - Nom_objet.méthode() pour appeler une méthode.
- Ces méthodes prennent une chaîne de caractères et retournent la même chaîne, mais entièrement en majuscule ou en minuscule.

```
element = "helium"
upper_element = element.upper()
print(upper_element)
```

```
HELIUM
```

Exercice: Lower:

- Créez une variable 'python' et assignez lui la phrase: "Bonjour Monde, J'APprends PYTHON"
- Utilisez la méthode lower pour transformer la phrase uniquement en lettres minuscules.
- Affichez le résultat dans la console.

python = "Bonjour Monde, J'APprends PYTHON"
lower_python = python.lower()
print(lower_python)

Activité

Nous allons créer un code qui lit des documents dans un dossier et cherche un mot spécifique. Si le mot est trouvé, le code affichera ”Match found” dans la console avec le document et le nombre de fois qu’il s’y retrouve.

On va tranquillement construire ce code durant l’atelier. Il faut d’abord ajouter les documents au dossier que nous avons créé pour cet atelier.

Maintenant, créons une variable qui s’appelle numberOfMatches et assignons lui la valeur 0.

```
numberOfMatches = 0
```

Listes

https://librarycarpentry.org/lc-python-intro/11-lists/index.html

Une liste contient plusieurs valeurs dans une seule structure.

- Scénario: Vous voulez noter plusieurs températures.
- Faire plusieurs calculs avec plusieurs centaines de variables qui s’appellent temperature_001, temperature_002, etc. reviendrait environ au même que faire les calculs manuellement.
- Pour créer une liste, vous devez assigner des crochets ([]) au nom de la liste. Vous pouvez la créer avec ou sans valeur.
    - Utilisez les crochets ([]) seuls pour représenter une liste sans valeur.
- On utilise les listes pour leur assigner plusieurs valeurs.
    - Les valeurs sont toutes enregistrées à l’intérieur des crochets [...].
    - On sépare chaque valeur par une virgule.
- Utilisez len pour connaître le nombre de valeur dans une liste.

```
temperatures = []
print(temperatures)
```

```
[]
```

```
temperatures = [17.3, 17.5, 17.7, 17.5, 17.6]
print(temperatures)
print("length of temperatures list", len(temperatures))
```

```
[17.3, 17.5, 17.7, 17.5, 17.6]
length of temperatures list 5
```

Utilisez l'index d'une des valeurs pour aller la chercher dans la liste.

- On commence à compter à partir de 0.
- On peut aussi utiliser cela avec des chaînes de caractères.

```
print(temperatures[0])
print(temperatures[4])
```

```
17.3
17.6
```

Exercice

- Essayez de renvoyer la dernière lettre d'une chaîne de caractères en utilisant len().

  Indice: Si Python commence à compter à zéro et que len() retourne le nombre de caractères dans une chaîne, quelle expression rendra la dernière lettre d'une chaîne de caractères?

  name[len(name) - 1]

- Essayez de renvoyer la dernière température en utilisant len().

  temperatures[len(temperatures) - 1]

Ajouter une valeur pour accroître la liste.

- Utilisez nom_liste.append(valeur) pour ajouter une valeur à la fin de la liste.

```
temperatures = [17.3, 17.5, 17.7, 17.5, 17.6]

temperatures.append(19.7)
print(temperatures)
```

```
[17.3, 17.5, 17.7, 17.5, 17.6, 19.7]
```

- Append est une méthode liée aux listes.

Conditionnels (si)

https://librarycarpentry.org/lc-python-intro/17-conditionals/index.html

Utilisez les phrases if pour contrôler si un bloc de code devrait être exécuté.

- Une phrase if contrôle si un bloc de code sera ou non exécuté.
- L'idée générale est:
  - Si x alors:
    - Faites A
  - Si y alors:
    - Faites B
  - A n'est exécuté que si les conditions de x sont remplies.
- Voici la structure:
  - If conditions :
    - Bloc de code
    - Il y a généralement un alinéa (équivalent à quatre espaces) avant le bloc de code.
  - Les conditions utilisent généralement <, >, ==, !=

```
#mass = 3.54
mass = 2.97
if mass > 3.0:
    print(mass, "is larger")
if mass < 3.0:
    print(mass, "is smaller")
```

```
2.97 is smaller
```

- Les alinéa sont importants, si vous ne les respectez pas, vous recevrez des erreurs.

```
firstName = "John"
 lastName = "Smith"
```

```
  File "<ipython-input-8-da09491f9801>", line 2
    lastName = "Smith"
    ^
IndentationError: unexpected indent
```

Activité:

Créez une phrase if qui évalue si la variable créée plus tôt (numberOfMatches) est supérieur à 0. Si le mot est trouvé au moins une fois, affichez 'match found' sur la console.

```
if numberOfMatches > 0:
    print("match found")
```

Utilisez elif pour ajouter des conditions supplémentaires et else pour exécuter du code si aucune des conditions n'est rencontrées.

- Il est possible d'offrir plusieurs conditions, chacune représentant son propre test.
- Utilisez elif (qui est un raccourci de else if (sinon) et la condition à remplir dans ce cas.
- Il faut toujours d'abord avoir un if.
- Une fois une condition remplie, le code arrête et ne regarde même pas les conditions suivantes. C'est pourquoi, ici, 9.7 n'est imprimé qu'une seule fois.
- Elif doit venir avant else.
-

```
mass = 4.9
if mass > 9.0:
    print(mass, "is huge")
elif mass > 3.0:
    print(mass, "is larger")
else:
    print(mass, "is smaller")
```

```
4.9 is larger
```

Les conditions sont testées une fois, dans l'ordre.

- Python passe à travers les phrases conditionnelles en ordre, les testant chacune à leur tour.
- L'ordre est très important!
- Pouvez-vous voir et proposer une solution à l'erreur ci-dessous?
-

```
grade = 85

if grade >= 70:
    print("Grade is C")
elif grade >= 80:
    print("Grade is B")
elif grade >= 90:
    print("Grade is A")
```

```
Grade is C
```

Boucles For

https://librarycarpentry.org/lc-python-intro/12-for-loops/index.html

Les boucles for exécutent certaines commandes une fois pour chaque valeur dans une collection.

- Faire des calculs sur toutes les valeurs d'une liste une à la fois ne fait pas beaucoup de sens.
- Disons que tu veux afficher chaque température dans la console. Tu pourrais faire ceci:

```
temperature_01 = 17.5
temperature_02 = 15.9
temperature_03 = 19.4
temperature_04 = 21.1

print(temperature_01)
print(temperature_02)
print(temperature_03)
print(temperature_04)
```

- En utilisant une boucle for, tu peux faire cela systématiquement.
- Une boucle for dit à Python d'exécuter un bloc de code une fois pour chaque valeur dans une liste, chaque caractère d'une chaîne de caractères, ou une valeur dans une collection.
- Pour chaque élément dans ce groupe, faites x.

```
for temperature in [17.5, 15.9, 19.4, 21.2]:
    print(temperature)
```

- Vous pouvez aussi utiliser une liste déjà construite.

```
temperatures = [17.5, 15.9, 19.4, 21.2]

for current_temperature in temperatures:
    print(current_temperature)
```

```
17.5
15.9
19.4
21.2
```

- Le résultat dans la console pour ces deux phrases devrait être le même.

La première ligne de la boucle doit se terminer par deux points et il doit y avoir des alinéa avant le bloc de code.

- Les alinéa sont essentiels, exactement comme pour les conditionnels.
    - Python utilise les alinéa pour montrer le bloc de code niché (lié à une phrase de code).

La boucle for est faites d'une collection, d'une variable de boucle, et d'un corps.

```
for number in [2, 3, 5]:
    print(number)
```

- La collection [2, 3, 5] est ce sur quoi la boucle est exécutée.
- Le corps - print(number) - spécifie ce qu'il faut faire pour chaque valeur de la collection. Le corps peut contenir plusieurs phrases.
- La variable de la boucle est ce qui change à chaque fois que la boucle revient au début.
    - La valeur du moment.

```
for number in [2,3,5]: # number is the loop variable. [2,3,5] is the collection, what the loop is running on.
    print(number) # The body specifies what to do for each value in the collection.
```

Vous pouvez utiliser range pour faire une boucle avec une séquence de chiffres.

- La fonction intégrée range produit une séquence de chiffres.
    - Ce n'est **pas** une liste. Les chiffres sont produits sur demande pour les boucles.
- range(N) contient les chiffres 0…N-1

```
for number in range(0,3):
    print(number)
```

```
0
1
2
```

Le modèle accumulateur transforme plusieurs valeurs en une seule valeur.

- Ceci est un modèle souvent utilisé en programmation:
    1. Initialiser la variable de l'accumulateur à 0, une chaîne ou une liste vides.
    2. Changer la variable avec les valeurs d'une collection.

```
# sum of the first 10 integers.

total = 0 # Accumulator variable
for number in range(10):
    total = total + (number + 1) # Remember, the range will start at 0.

print(total)
```

```
55
```

- Lisez total = total + (number + 1) comme ceci:
    - Ajoutez 1 à la valeur du nombre de la variable de la boucle.
    - Ajoutez cette valeur à la valeur totale de la variable de l'accumulateur.
    - Enregistrez cette valeur dans le total pour remplacer la valeur qui y était.
    -
- Il faut ajouter +1 parce que range produit les chiffres 0-9 et non 1-10.

Activité

- Vous pouvez utiliser les boucles et les conditionnels pour accroître les valeurs des variables. Dans l'activité créez une boucle for qui lie chaque ligne dans la variable reader. Ensuite transformez chaque caractère de cette ligne en lettres minuscules.
- Créez une phrase si qui cherche le mot sleep dans cette ligne.
- Si le mot est trouvé, ajoutez 1 à la variable numberOfMatches, en utilisant le modèle accumulateur.

```
for line in reader:
    line = line.lower()
    if "sleep" in line:
        numberOfMatches = numberOfMatches + 1
```

Utilisez des bibliothèques

https://librarycarpentry.org/lc-python-intro/06-libraries/index.html

La grande force des langages de programmation se retrouve dans les bibliothèques.

- Une bibliothèque est une collection de modules qui contiennent des fonctions que tous peuvent utiliser.
    - Le contenu d'une bibliothèque devrait être lié ou suivre une thématique, mais il n'y a aucune manière de forcer cette règle.
- La bibliothèque de base de Python contient plusieurs modules.
- Certaines bibliothèques supplémentaires utilisées couramment viennent avec Anaconda.
- Les bibliothèques sauvent beaucoup de temps, en nous permettant de réutiliser le code des autres.

Un programme doit d'abord importer une bibliothèque ou un module avant de l'utiliser.

- Utilisez import pour télécharger un module dans la mémoire du programme.
- Vous devez ensuite utiliser le nom du module pour l'utiliser: nom_module.nom_chose.
    - Avec Python, le point signifie 'faire partie de'
- Pour l'activité, nous allons importer la bibliothèque os
    - OS: un module qui interagit avec le système d'exploitation de l'ordinateur.
    - Par exemple: lire ou écrire dans un document ou accéder à un emplacement.

```
# import a library
import os
```

```
currentDir = os.getcwd()
print(currentDir)
```

```
C:\Users\cturp1\Desktop\PythonWorkshop
```

Activité

Regardons le reste du code, en utilisant le code que nous avons déjà écrit.

```
# import a library
import os

## Script:

currentDir = os.getcwd()
filesDir = currentDir + "\\FairyTales\\"

#Python method listdir() returns a list of files / folders in the directory given.
for filename in os.listdir(filesDir):

    numberOfMatches = 0
    currentFile = filesDir + filename

    if filename != ".ipynb_checkpoints":
        myFile = open(currentFile, "r", encoding = "utf-8", errors = "ignore")
        reader = myFile.readlines()
        for line in reader:
            line = line.lower()
            if 'sleep' in line:
                numberOfMatches = numberOfMatches + 1

    if numberOfMatches > 0:
        print("match found", numberOfMatches, "times in:", filename)
```