Chapitre1

Install de **Visual Studio Code (ide de dev)**

**Git (j’ai gitbash)**

**Github cloud, créer nouveau projet dans mon github et cloner sur ma machine**

**Cloner aussi leur projet a coté, copier tout sauf le .git dans mon dossier**

**Leur projet se presente sous la forme de repertoire pour faire les exos et de repertoires corrections**

**print("hello, world!")**

les variables

**print(f**

**nom = "Dupont"**

**prenom = "Jean"**

**age = 30**

**print(f"Bonjour, je m'appelle {prenom} {nom} et j'ai {age} ans.")**

**type()**  permet de déterminer le type d’une variable.

**Rappel :**

**Listes[,] maiss ensuite si on appelle une méthode on utilise ()**

**Tuples(,) c’est une liste non modifiable**

**Dictionnaires{cle : valeur, cle : valeur }**

Listes[]

**plateformes\_sociales = ["Facebook", "Instagram", "Snapchat", "Twitter"]**

**affichage**

plateformes\_sociales[0]

"Facebook"

**modif**

plateformes\_sociales[2] = "LinkedIn"

**ajout**

plateformes\_sociales.append("TikTok")

**suppression**

plateformes\_sociales.remove("Snapchat")

**longueur**

len(plateformes\_sociales)

**tri**

plateformes\_sociales.sort()

**autres**

|  |  |
| --- | --- |
| extend() | Ajoute plusieurs éléments à la fin de la liste. |
| insert() | Insère un élément à une position donnée dans la liste. |
| pop() | Supprime et renvoie l'élément à une position donnée de la liste, ou le dernier élément si aucun indice n'est spécifié. |
| index() | Renvoie la première occurrence de l'élément spécifié dans la liste. |
| count() | Renvoie le nombre d'occurrences de l'élément spécifié dans la liste. |
| reverse() | Inverse l'ordre des éléments de la liste. |

**les tuples (c’est des listes mais on utilise les () au lieu des [] et les tuples ne sont pas modifiables**

plateformes\_sociales\_tuple = ("Facebook", "Instagram", "TikTok", "Twitter")

### ****Trouvez un élément avec in****

Pour savoir si un élément est présent dans une liste ou un tuple, on peut utiliser l'opérateur in  . Cet opérateur retourne True  si l'élément est présent dans la séquence, et False  sinon.

>>> nombres = [1,2,3,4,5]

>>> 5 in nombres

True

>>> 8 in nombres

False

### ****dictionnaire = {liste de clé : valeur} séparées par des ,****

### ****note : un dict peut contenir des valeurs sous forme de liste****

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"]**

**}**

### ****Accédez à une valeur dans un dictionnaire (on utilise ici les [] )****

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

#### **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

**infos\_labradoodle['nom\_scientifique'] = "Canis lupus familiaris"**

#### **Supprimez une paire clé-valeur**

Pour supprimer une paire clé-valeur, vous pouvez utiliser le mot-clé  del

>>> del infos\_labradoodle["origine"]

Voici encore quelques méthodes couramment utilisées pour manipuler des dictionnaires :

|  |  |
| --- | --- |
| keys() | ​​Retourne une vue sur les clés du dictionnaire. |
| values() | Retourne une vue sur les valeurs du dictionnaire. |
| items() | Retourne une vue sur les couples (clé, valeur) du dictionnaire. |
| get(clé) | Retourne la valeur associée à la clé spécifiée. Si la clé n'est pas présente dans le dictionnaire, retourne la valeur None  . |
| pop(clé) | Supprime la clé spécifiée et retourne la valeur associée. Si la clé n'est pas présente dans le dictionnaire, retourne la valeur None  . |
| clear() | Supprime tous les éléments du dictionnaire. |

#### **Vérifiez l’existence d’une clé spécifique**

Vous pouvez utiliser le mot-clé in

>>> "poids" in infos\_labradoodle

True

#1. \*\*Créez\*\* un \*\*dictionnaire\*\* appelé `fruits` avec les clés `"pomme"`, `"banane"` et `"orange"`, et respectivement les valeurs `"rouge"`, `"jaune"` et `"orange"`.

fruits = {"pomme":"rouge", "banane":"jaune", "orange":"orange"}

print(fruits)

#2. \*\*Ajoutez\*\* la clé `"kiwi"` avec la valeur `"vert"` au dictionnaire `fruits`.

fruits["kiwi"] = "vert"

print(fruits)

#3. \*\*Accédez\*\* à la valeur correspondant à la clé `"banane"` et stockez-la dans une variable appelée `couleur\_banane`.

couleur\_banane = fruits["banane"]

print (couleur\_banane)

#4. \*\*Modifiez\*\* la valeur associée à la clé `"pomme"` pour `"vert"`.

fruits["pomme"] = "vert"

print(fruits)

#5. \*\*Supprimez\*\* la clé `"banane"` du dictionnaire `fruits`.

del fruits["banane"]

print(fruits)

#6. \*\*Affichez\*\* les clés restantes dans le dictionnaire.

print(fruits.keys())

print(fruits.values())

print(fruits.items())

**Chapitre 2 :**

**Les conditions**

Les **instructions if/elif/else** vous permettent de définir des conditions multiples.

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 !")

**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).

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 !")

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

Assigner à une variable :

nombre\_de\_sieges = 30

Tester si nombre\_de\_sieges est égal à 30 :

nombre\_de\_sieges == 30

### ****les match cases****

fruit = "pomme"

match fruit:

case "pomme":

print("J'aime les pommes !")

case "banane":

print("Je n'aime pas les bananes.")

case "orange":

print("Les oranges sont bonnes pour la santé.")

case \_:

print("Je ne connais pas ce fruit.")

les boucles

for

races\_de\_chien = ["golden retriever", "chihuahua", "terrier", "carlin"]

for chien in races\_de\_chien:

print(chien)

range

for x in range(5):

print(x)

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

La fonction range() est réglée sur 0 par défaut pour la valeur de début. Vous pouvez la modifier en ajoutant un autre nombre entier comme ça : range(4, 10) !

Cette plage renvoie les valeurs de 4 à 9 (sans inclure 10).

While

capacite\_maximale = 10

capacite\_actuelle = 3

while capacite\_actuelle < capacite\_maximale:

capacite\_actuelle += 1

break (sortie de la boucle)

for i in range(10):

if i == 5:

break

print(i)

Continue (passe a l’iteration suivante sans executer le code en dessous)

liste = [1, 2, 3, 4, 5]

# Boucle for sur la liste

for element in liste:

if element == 3:

# Si l'élément vaut 3, on passe à l'itération suivante sans exécuter le reste du code

continue

# Dans tous les autres cas, on exécute le reste du code

print(element)

Si un élément vaut 3, l'instruction  continue  est exécutée, ce qui fait passer directement à l'itération suivante sans exécuter le reste du code présent dans la boucle pour cet élément. Ainsi, la sortie de ce code nous donnera : 1 2 4 5

Correction exercice

def main():

    # Récupérer la saisie de l'utilisateur

    liste = input("Saisissez une liste de nombres séparés par des virgules : ")

    # Séparer l'ensemble des nombres et les insérer dans une liste

    liste = liste.split(",")

    # Afficher la liste des nombres

    print("Liste des nombres :", liste)

    # Calculer la somme des nombres

    somme = 0

    for nombre in liste:

        somme += int(nombre)

    print("Somme des nombres :", somme)

    # Effectuer la moyenne à l'aide de la somme des nombre

    moyenne = somme / len(liste)

    print("Moyenne des nombres :", moyenne)

    # Trouver le nombre d'entier supérieur à la moyenne

    nombre\_sup\_moyenne = 0

    for nombre in liste:

        if int(nombre) > moyenne:

            nombre\_sup\_moyenne += 1

    print("Nombre de nombres supérieurs à la moyenne :", nombre\_sup\_moyenne)

    # Trouver le nombre d'entier pair

    nombre\_pairs = 0

    i = 0

    while i < len(liste):

        if int(liste[i]) % 2 == 0:

            nombre\_pairs += 1

        i += 1

    print("Nombre de nombres pairs :", nombre\_pairs)

# Ne touchez pas le code ci-dessous

if \_\_name\_\_ == "\_\_main\_\_":

    main()

les fonctions

sans parametres

def afficher\_message():

print("Bonjour, comment ça va ?")

avec parametres

def afficher\_nom\_prenom(nom, prenom):

print("Nom :", nom)

print("Prénom :", prenom)

avec return

il faut que le fonction renvoi dans une var. ici resultat. Pour qu’elle puisse renvoyer le resultat quand on l’appelle.

Mais la var resultat n’est pas connue en dehors de la fonction.

Juste son contenu sera renvoyé quand elle est appelée

Somme contient le retour de la fonction, donc le contenu de résultat

def main():

   somme = calculer\_somme(2, 3)

   print(somme) #Ce print affichera 5

def calculer\_somme(a, b):

  resultat = a + b

  return resultat

Il est possible de retourner **plusieurs** valeurs en les **séparant** par des **virgules** dans l'instruction de retour de la fonction. Les valeurs retournées seront automatiquement regroupées dans un **tuple**.

Correction

# Définition de la fonction salaire\_mensuel

def salaire\_mensuel(salaire\_annuel):

    return salaire\_annuel / 12

# Définition de la fonction salaire\_hebdomadaire

def salaire\_hebdomadaire(salaire\_mensuel):

    return salaire\_mensuel / 4

# Définition de la fonction salaire\_horaire

def salaire\_horaire(salaire\_hebdomadaire, heures\_travaillees):

    return salaire\_hebdomadaire / heures\_travaillees

def main():

    # Demande à l'utilisateur de saisir son salaire annuel

    salaire\_annuel = float(input("Entrez votre salaire annuel : "))

    # Demande à l'utilisateur de saisir le nombre d'heures travaillées par semaine

    heures\_travaillees = float(

        input("Entrez le nombre d'heures travaillées par semaine : "))

    # Calcul du salaire horaire

    mensuel = salaire\_mensuel(salaire\_annuel)

    hebdomadaire = salaire\_hebdomadaire(mensuel)

    horaire = salaire\_horaire(hebdomadaire, heures\_travaillees)

    # Affichage du résultat

    print("Votre salaire horaire est de", horaire, "euros.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

commentaires

# ceci est un commentaire

les **docstrings** sont des chaînes de documentation qui sont utilisées pour **documenter** les fonctions,

on utilise **3 guillemets**

def somme(a, b):

**"""**

Cette fonction calcule la somme de deux nombres et retourne le résultat.

Parameters:

a (int): le premier nombre

b (int): le deuxième nombre

Returns:

int: la somme de a et b

**"""**

return a + b

Lorsque vous appelez cette fonction dans l'interpréteur Python, vous pouvez utiliser la fonction  help()  pour afficher la docstring, voici le résultat :

>>> help(somme)

somme(a, b)

Cette fonction calcule la somme de deux nombres et retourne le résultat.

Parameters:

a (int): le premier nombre

b (int): le deuxième nombre

Returns:

int: la somme de a et b

gestion d’erreurs et d’exceptions

bloc  try/except

while True:

try:

x = int(input("Entrez un nombre entier : "))

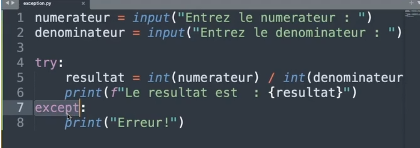
break

except ValueError:

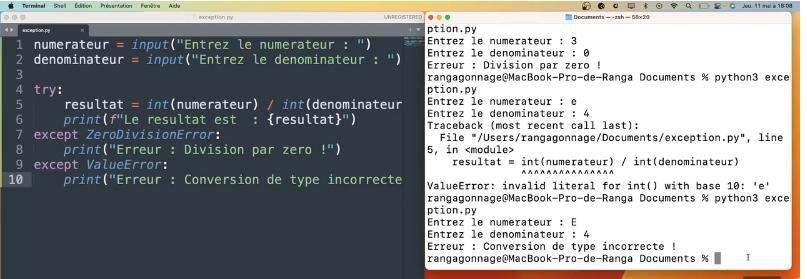
print("Oops ! Ce n'est pas un nombre entier. Essayez encore...")

Dans ce code, le bloc  try  contient le code qui peut causer une erreur, c'est-à-dire la conversion de la saisie utilisateur en un entier. Si une erreur se produit, le bloc  except  est exécuté et affiche un message d'erreur personnalisé. Le programme revient ensuite au début de la boucle  while  et demande à l'utilisateur de saisir un nouveau nombre.

On peut utiliser except sans specifier le type d’erreur. Le except prendra en compte tooutes les erreurs possibles, mais sans donner de détail



Il est préférable de lever le type d’erreur renvoyé par le code (dans le terminal) et d’ajouter un except pour chaque type d’erreur rencontrée



Chapitre 3 modules et packages

### ****Importez un module****

si vous avez un fichier nommé mon\_module.py  qui contient une fonction nommée ma\_fonction  , vous pouvez l'importer ainsi :

**import mon\_module**

**resultat = mon\_module.ma\_fonction()**

Une fois le module importé dans votre nouveau fichier grâce à la commande import  , vous pouvez désormais utiliser la fonction ma\_fonction()  . Cependant, n'oubliez pas de spécifier le nom du module avant d'appeler la fonction.

Vous pouvez également importer des éléments **spécifiques** d'un module, en utilisant la syntaxe  from mon\_module import ma\_fonction   . Cela vous permet d'utiliser directement la fonction ma\_fonction  sans avoir à spécifier le nom du module.

**from mon\_module import ma\_fonction**

**resultat = ma\_fonction()**

### ****Créez et importez un package****

Un package en Python est simplement un **dossier** contenant un **ensemble de modules** Python. Les packages permettent **d'organiser** votre code en sous-dossiers, et de créer des hiérarchies de modules.

Pour créer un package, vous devez simplement créer un dossier contenant un fichier nommé \_\_init\_\_.py  . Ce fichier est utilisé pour initialiser le package, et peut contenir du code d'initialisation si nécessaire.

Pour utiliser un module d'un package, vous devez spécifier le **nom** du package dans l'import. Par exemple, si vous avez un package nommé mon\_package  qui contient un module nommé mon\_module  , vous pouvez l'importer ainsi :

import mon\_package.mon\_module

resultat = mon\_package.mon\_module.ma\_fonction()

Tout comme pour les modules, vous pouvez importer les éléments spécifiques du module d'un package en utilisant la syntaxe  from mon\_package.mon\_module import ma\_fonction   .

De nombreux packages populaires sont disponibles sur des dépôts en ligne tels que [**PyPI**](https://pypi.org), et peuvent être **facilement** installés à l'aide d'un gestionnaire de packages tel que pip  .

Voici quelques exemples de packages populaires, et leurs fonctions :

* [**Requests**](https://fr.python-requests.org/en/latest/) : un package HTTP élégant et simple pour Python. Fréquemment utilisé pour les appels d’interface REST ;
* [**Beautiful Soup**](https://www.crummy.com/software/BeautifulSoup/bs4/doc/) (ressource en anglais) : un package pour récupérer des données de fichiers HTML et XML ;
* [**Pandas**](https://pandas.pydata.org/) (ressource en anglais) : un outil open source rapide, puissant et accessible pour l’analyse et la manipulation de données.

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

**ETL** signifie **extraction**, **transformation** et **chargement** (Extract, Transform, Load, en anglais)

L’exercice extrait du code d’une page html

La il faut voir le code utilisé…p3c2

P3c3

Lecture ecriture de données dans des fichiers txt, csv

Avec open()

### ****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 (sans 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](https://docs.python.org/fr/3/library/csv.html) 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 puisse 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

Bien plus utile, n’est-ce pas ?

Il y a un exemple de parsing d’une page web et ecriture dans csv mais il faut regarder la video dans la formation et le code est dans

C:\Users\Laurent\Documents\Formation\openclassroom\_python\_les\_bases\7168871-apprenez-les-bases-du-langage-python\Scripts\_Apprenez les bases de Python 4

Dans le p3c3 l’exo lit un csv, transform les datas et ecrit dans un autre csv

Une correction du quiz qui lit html et ecrit csv

from bs4 import BeautifulSoup

import requests

URL = "https://quiz-partie3.oc"

def extract\_data(url):

response = requests.get(url)

html = response.content

return html

def get\_title(soup):

return soup.title.string

def get\_all\_product\_info(soup, products):

for table in soup.find\_all('table'):

for tr in table.find\_all('tr')[1:]:

td = tr.find\_all('td')

name = td[0].get\_text()

description = td[1].get\_text()

price = td[2].get\_text()

quantity = td[3].get\_text()

product = {

'nom': name,

'description': description,

'prix': price,

'quantite': quantity

}

products.append(product)

def convert\_dollar\_to\_euro(products):

change\_rate = 0.8

for product in products:

price\_euro = product['prix'].replace('€', '')

price\_euro = float(price\_euro)

price\_dollar = price\_euro \* change\_rate

product['prix'] = str(round(price\_dollar, 2)) + '$'

def load\_data(soup, products):

title = get\_title(soup)

with open(f'{title}.csv', mode='w', newline='') as file:

fieldnames = ['nom', 'description', 'prix', 'quantite']

writer = csv.DictWriter(file, fieldnames=fieldnames)

writer.writeheader()

for product in products:

writer.writerow(product)

if \_\_name\_\_ == '\_\_main\_\_':

html = extract\_data(url)

soup = Beautiful\_Soup(html, "html.parser")

products = []

get\_all\_product\_info(soup, products)

convert\_dollar\_to\_euro(products)

load\_data(soup, products)