

python

[\[🔗 link\]](#) — [\[Bonus\] - Plus de 120 exercices d'algorithmie en Python - YouTube - Foxxpy - Mathématiques et algorithmie](#)

.python basics

- [\[🔗 link\]](#) — 5 sites à coder pour prouver vos compétences - YouTube - Docstring
- [\[🔗 link\]](#) — Créer et héberger un site web avec Python : Formation Complète - YouTube - Docstring
- [\[🔗 link\]](#) — J'ai codé un outil de quiz illimités avec une IA - YouTube - [Docstring](#)
- [\[🔗 link\]](#) — Mini Python Project Tutorial - Password Generator - YouTube - [Tech With Tim](#)
- [\[🔗 link\]](#) — freeCodeCamp.org - YouTube -
- [\[🔗 link\]](#) — [mCoding](#) - YouTube -
- [\[🔗 link\]](#) — This is How I Produce My Content - YouTube - [NeuralNine](#)
- [\[🔗 link\]](#) — Patrick Loeber - YouTube - Patrick Loeber
- [\[🔗 link\]](#) — ArjanCodes - YouTube -
- [\[🔗 link\]](#) — PYTHON - Envoi de messages automatisés basés sur les noms de pays | Tutoriel de codage - YouTube - **Hex CG**
- [\[🔗 link\]](#) — Gestion des fichiers — Python 3.X
- [\[🔗 link\]](#) — Système Authentification Avec Python Tkinter
- [\[🔗 link\]](#) — Système de Gestion des Factures Avec Python Django - YouTube - Donald Programmeur
- [\[🔗 link\]](#) — Déployer une application web Django sur PythonAnywhere [GRATUIT] | Astuces Django - YouTube - Donald Programmeur
- [\[🔗 link\]](#) — comment creer un raccourci clavier pour une tache d'automatisation - Recherche Google
- [\[🔗 link\]](#) — Comprendre Python en 5 minutes
- [\[🔗 link\]](#) — Google Scholar

- [[link](#)] — 3 PYTHON AUTOMATION PROJECTS FOR BEGINNERS - YouTube - Internet Made Coder
- [[link](#)] — Create A Python API in 12 Minutes - YouTube - Tech With Tim
- [[link](#)] — interpreted and compiled language - Recherche Google



la casse désigne **le fait de distinguer les lettres capitales des lettres minuscules**

pour executer un fichier python en ligne de commande `python teste.py`

▼ TIPS

- [[link](#)] — Impossible de supprimer un fichier qui est en cours d'utilisation sous Windows 10 ! - Rene.E Laboratory

F2 pour renommer rapidement un variable

shift + alt + bas pour dupliquer

shift alt f pour indenter tous le code avec lextention pretier

alt + souris pour select plusieurs lignes

cntrl + / commenter un code pareil pour decommenter

selectionner + cntrl d pour selectionner toute les occurance de cette eleement

code . pour ouvrir un projet sur vcs

code ./projet pour ouvrir un projet qui est dans un dossier sur vcs



Python est fortement typés. (on ne peut pas manipuler 2 variables avec 2 type differente ex un entier = une chaîne de caractère il faut dans ce cas utiliser les fonction int ou str ...) Un entier ne peut être traité comme une chaîne sans conversion explicite les types utilisés dans le code source (fonction, variable, etc.) sont vérifiés au moment de la compilation.
langage faiblement typé Un langage dans lequel les types peuvent être ignorés

Typage Fort :

- **Typage fort signifie que les types des objets sont strictement appliqués.** Une fois qu'un objet a été attribué à un certain type, il ne peut pas être utilisé comme s'il était d'un autre type sans conversion explicite.
- Par exemple, vous ne pouvez pas additionner un entier et une chaîne de caractères sans effectuer une conversion explicite.

```
a = 5
```

```
b = "10"
```

```
result = a + b # va generer une erreur
```

Typage Dynamique :

- **Typage dynamique signifie que les types des objets sont déterminés à l'exécution.** Vous n'avez pas besoin de déclarer le type d'une variable explicitement lors de sa création, et le type d'une variable peut changer au cours de l'exécution du programme.
- Cela donne à Python une grande flexibilité, mais cela nécessite également une attention particulière pour éviter des erreurs de type.

```
a = 5
```

```
print(type(a)) # <class 'int'>
```

Langage interprété : (python , php ..)

1. **Interprétation en temps réel :** Dans un langage interprété, le code source est exécuté ligne par ligne ou instruction par instruction en temps réel par un interpréteur. Cela signifie que le code source est lu

et exécuté directement sans passer par une étape de compilation préalable.

Langage compilé : (c , java ..)

1. Compilation préalable : Dans un langage compilé, le code source est d'abord passé par un compilateur qui traduit l'ensemble du code en code machine ou en un autre langage intermédiaire. Le code résultant est ensuite exécuté.

En Python, la mémoire est gérée dans un espace appelé tas privé (pile d'espace mémoire disponible) "private heap space", où l'interpréteur Python stocke tous ses objets et structures de données. Le gestionnaire de mémoire est responsable de l'allocation et de la libération de la mémoire pour l'interpréteur Python. Le gestionnaire de mémoire Python effectue également la collecte automatique des déchets, qui récupère la mémoire inutilisée et la rend disponible pour d'autres objets python est un langage typé dynamiquement, c'est-à-dire qu'**on peut changer le type d'une variable**. A contrario, un langage typé statiquement force à définir le type des variables et à le conserver au cours de la vie de la variable



avec l'extension **python** on peut créer un fichier ipynb (pour essayer des trucs et commenter)



codesnap : extension visuelle pour faire de belles captures d'écran du code


python ne possède aucun type de données pour les caractères, ils sont traités comme des chaînes de caractères (String).

int(string, [base]) :


la fonction convertit la chaîne donnée en nombre décimal, ici la base est facultative où vous pouvez donner n'importe quel nombre, quelques exemples sont Mais veuillez noter que la chaîne que vous donnez doit être un nombre dans cette base





Links

[ LINK] — Let's build GPT: from scratch, in code, spelled out. - YouTube
- Andrej Karpathy

[ LINK] — Certification Catalog - Linux Foundation - Training

[ LINK] — Gestion de la mémoire en Python – StackLima

 [ LINK] — Le langage typé fort ou faible, interprété ou compilé, une définition

Les_commandes_indispensables_du_Terminal.pdf

Les_types_natifs.pdf

Interagir_avec_lutilisateur.pdf

Liste_des_mthodes_de_chanes_de_caractres.pdf

Le_formatage_des_chaines_de_caracteres.pdf

Les_types_natifs.pdf



les variables

```
1. >>> a = [1, 2, 3]
2. >>> b = a
3. >>> b += [4]
4. >>> print(a)
5. [1, 2, 3, 4]
```

Dans l'exemple ci-dessus, les noms `a` et `b` pointent vers la même liste en mémoire.

Avec l'affectation simple, on associe **une** valeur à **une** variable :

```
a = 5
```

Avec l'affectation parallèle, on associe **plusieurs** valeurs à **plusieurs** variables :

```
a, b = 5, 10
```

Avec l'affectation multiple, on associe **une** valeur à **plusieurs** variables :

```
a = b = 5
```

une variable **ne peut pas commencer par un nombre**.

tirets du bas pour séparer les mots

le nom d'une variable ne peut pas contenir certains caractères spéciaux

Un nom de variable ne peut également pas contenir d'espaces.

BOOL

On utilisera beaucoup les booléens par la suite car ils vont permettre à nos scripts de prendre des décisions !

En fonction du résultat d'une expression, qui retournera `True` ou `False`, vous pourrez ainsi orienter votre

programme dans la direction souhaitée.

******pas besoin donc de spécifier le type de la variable à créer. Python est suffisamment intelligent pour le déduire lui-même (grâce par exemple aux guillemets pour les chaînes de caractères).mais on a des constructeur d'objets

- `str()`
- `int()`
- `float()`
- `bool()`

Ces classes ne sont pas très utiles pour créer des objets.

Mais elles sont très pratiques pour *convertir* un objet d'un type à un autre !

les strings :

```
variable = "guillemets"
```

```
longue_phrase = '''Bonjour,  
bienvenue sur Docstring,  
'''  
nombre = 15  
resultat = "Le nombre est " + str(nombre)  
EXPLICATION
```

Dans Python, on ne peut pas concaténer des variables de différents types

`help(str.center)` pour afficher l'aide de la méthode `center`)

`len(chaine)` : Renvoie la longueur de la chaîne de caractères.

`chaine.upper()` : Renvoie une copie de la chaîne en majuscules.

`chaine.lower()` : Renvoie une copie de la chaîne en minuscules.

`chaine.capitalize()` : Renvoie une copie de la chaîne avec la première lettre en majuscule et les autres en minuscules.

`chaine.title()` : Renvoie une copie de la chaîne avec la première lettre de chaque mot en majuscule.

`chaine.strip()` : Renvoie une copie de la chaîne avec les espaces et les caractères de saut de ligne supprimés des bords.

`chaine.replace(ancien, nouveau)` : Renvoie une copie de la chaîne avec toutes les occurrences de l'ancien texte remplacées par le nouveau texte.

`chaine.split(separateur)` : Divise la chaîne en une liste de sous-chaînes en utilisant le séparateur spécifié.

`chaine.join(iterable)` : Concatène les éléments d'un iterable (par exemple, une liste) en utilisant la chaîne comme séparateur.

`chaine.startswith(prefixe)` : Vérifie si la chaîne commence par le préfixe spécifié et renvoie un booléen.

`chaine.endswith(suffixe)` : Vérifie si la chaîne se termine par le suffixe spécifié et renvoie un booléen.

`chaine.find(sous_chaine)` : Recherche la première

re occurrence de la sous-chaîne dans la chaîne et renvoie l'indice de début, ou -1 si la sous-chaîne n'est pas trouvée.

`chaîne.count(sous_chaine)` : Compte le nombre d'occurrences de la sous-chaîne dans la chaîne.

`chaîne.isalpha()` : Vérifie si la chaîne est composée uniquement de lettres alphabétiques et renvoie un booléen.

`chaîne.isnumeric()` : Vérifie si la chaîne est composée uniquement de chiffres et renvoie un booléen.

`chaîne.isalnum()` : Vérifie si la chaîne est composée de lettres et/ou de chiffres et renvoie un booléen.

les slices :

```
word = 'Python'
```

```
word[0:2] # caractères commençant de la position 0 (incluse) à 2 (exclue)  
-->'Py'
```

```
word[2:5] # caractères commençant de la position 2 (incluse) à 5 (exclue)  
-->'tho'
```

```
word[:2] # caractère commençant du début à la position 2 (exclu)  
-->'Py'
```

```
word[4:] # caractères commençant de la position 4 (incluse) jusqu'à la fin  
-->'on'
```

```
word[-2:] # caractères commençant de l'avant-dernier (inclus) à la fin  
-->'on'
```

```

word[::2] #caracteres du debut a la fin avec
pas de 2
-->'pto'
word[1:4:] #caracteres de la 1ere position a la
4eme avec pas de pas
print(word)
-->'yth'
word[1:-2:2] #caracteres de la 1ere position
a l'avant dernière(exclu) avec pas de 2
print(word)
-->'yh'
word[::-1] # on inverse l'ordre de la liste (on
recupere tous les elements partant du debut
# ju
squa la fin en commençant de la dernière posis
ition)
print(word)
-->'nohtyp'
word[1:-1] # on exclus le premier et le dernier
element
-->'ytho'
word[:2] + word[2:]
-->'Python'
word[:4] + word[4:]
-->'Python'

```

```

chaîne = "Pierre, Julien, Anne, Marie, Lucien"

```

```

chaîne_liste = chaîne.split(", ") #Sépare la c
haîne de caractères et sur celle en argument
et retourne une liste

```

```

liste_en_ordre = sorted(chaîne_liste) # trier
la liste

```

```

chaîne_en_ordre = ", ".join(liste_en_ordre) #
joindre la liste trier avec le caractère (, )

```

```

texte = "Ligne 1\nLigne 2\nLigne 3"

```

```
# Utilisez splitlines() pour diviser le texte
en lignes
lignes = texte.splitlines()
```

```
# Affichez chaque ligne individuellement
for ligne in lignes:
    print(ligne)
```

La méthode `splitlines()` en Python est une méthode de la classe de chaînes de caractères (`str`) qui permet de diviser une chaîne de caractères en une liste de sous-chaînes en utilisant les séquences de saut de ligne, comme `"\n"` (nouvelle ligne) ou `"\r\n"` (retour chariot suivi d'une nouvelle ligne), comme délimiteurs. Cette méthode est utile lorsque vous avez une chaîne de caractères contenant plusieurs lignes de texte et que vous souhaitez les diviser en lignes individuelles.

```
texte = "Ceci est un exemple de texte à diviser
en mots"
```

```
# Utilisez split() pour diviser le texte en mots
mots = texte.split()
```

```
# Affichez chaque mot individuellement
for mot in mots:
    print(mot)
```

```
Ceci
est
un
exemple
de
texte
à
diviser
```

```

en
mots

texte = "Un,exemple;de:texte"

# Utilisez split() avec un délimiteur personnalisé
elements = texte.split(",") # Diviser en utilisant la virgule comme délimiteur

# Affichez chaque élément individuellement
for element in elements:
    print(element)
Un
exemple;de:texte

strip()
Supprime les caractères spécifiés du début et de la fin de la chaîne

prenom = " .Marc,"
prenom_clean = prenom.strip(" .,")
print(prenom_clean) #Marc

La fonction isinstance() vérifie si l'objet (premier argument) est une instance ou une sous-classe de la classe classinfo (deuxième argument).

words = ['D', 'A', 'C', 'B']
numbers = [1, 2, 3, 4, 2, 5]

# check if numbers is instance of list
result = isinstance(numbers, list)
print(result) #True
result = isinstance(numbers, int)
print(number, 'instance of int?', result) # 5 i

```

```
nstance of int? True
```

```
isdigit()
```

```
user_input = input("Entrez quelque chose : ")
```

```
if user_input.isdigit():  
    print("L'entrée de l'utilisateur n'est pas  
une chaîne de caractères.")  
else:  
    print("L'entrée de l'utilisateur est une c  
haîne de caractères.")
```

```
methode replace
```

```
phrase = "Bonjour tout le monde."  
nouvelle_phrase = phrase.replace("Bonjour", "B  
onsoir")  
print(nouvelle_phrase)
```

```
methode count
```

```
lettre_a_chercher = "o"  
phrase = "Bonjour tout le monde"  
resultat = phrase.count(lettre_a_chercher)  
print(resultat)
```

```
methode Len()
```

```
longueur = len("jugo")  
print(longueur) -->4
```

la méthode find() recherche la première occurrence de la valeur spécifiée et renvoie -1 si la valeur n'est pas trouvée.

La méthode find() est presque pareil à la méthode index(), la seule différence est que la méthode index() lève une exception si la valeur

n'est pas trouvée

```
string.find(value, start, end)
```

```
"1, 2 ,3 ,4 ,5".split(" ,")
```

```
--> ['1', '2', '3', '4', '5']
```

```
".".join("1, 2 ,3 ,4 ,5".split(" ,"))
```

```
--> 1.2.3.4.5
```

```
" bonjour ".strip(" ujo") #supprime les caractères spécifiés en commençant par la gauche si il trouve par le caractère comme ici le b il va à la droite
```

```
---> 'bon'
```

la casse désigne le fait de distinguer les lettres capitales des lettres minuscules

```
"2".zfill(4)
```

```
'0002'
```



formatage des chaînes de caractères

au début

Pour concaténer des éléments ensemble on peut utiliser l'opérateur +
1/La concaténation peut se faire très simplement avec l'opérateur +

```
protocole = "https://"
```

```
nom_du_site = "Docstring"
```

```
extension = "fr"
```

```
url = protocole + "www." + nom_du_site + "." + extension
```

```
>>> age = 26
>>> phrase = "J'ai " + str(age) + " ans."
"J'ai 26 ans." #il faut convertir la variable age en chaîne
```

Il existe un autre moyen d'insérer des objets dans une chaîne, c'est d'utiliser la méthode `format` qui permet d'insérer directement le type des données insérées. Pour ce faire, on utilise l'opérateur modulo (%) et on s'assure que le type de la donnée correspond à celui qu'on insère dans la chaîne de caractères.

qui est devenu une méthode obsolète

```
age = 26
phrase = "J'ai %d ans." %age # "J'ai 26 ans."
phrase = "J'ai %f ans." %age # "J'ai 26.000000 ans."
En utilisant %d on se retrouve avec un nombre entier.
En utilisant %f, la variable age, qui est au départ un nombre entier, est converti en chaîne de caractères sous forme de nombre décimal.
```

2/méthode format:

Pour éviter les erreurs de type et insérer des objets dans une chaîne, on utilise la méthode `format`. Pour cela, on passe le prénom et l'âge en argument de la méthode `format`.

```
prenom = "Pierre"
age = 26
phrase = "Je m'appelle {} et j'ai {} ans.".format(prenom, age)
```

le nombre d'éléments passés à la méthode `format` doit être égal au nombre de accolades dans la chaîne.

```
age_de_l'utilisateur = 26
phrase = "J'ai {age} ans".format(age=age_de_l'utilisateur)
```

```
prenom = "Pierre"
age = 26
langage = "Python"
phrase = "Je m'appelle {0} et j'ai {1} ans. {0} apprend le {2}."
# Je m'appelle Pierre et j'ai 26 ans. Pierre apprend le Python
```

```
protocole = "https://"
nom_du_site = "Docstring"
extension = ".fr"
```

Avec l'opérateur +


```
url = protocole + "www." + nom_du_site + "." + extension
```

```
# Avec la méthode format
```

```
url = "{}www.{}.{}".format(protocole, nom_du_site, exten
```

```
url = "{protocole}www.{domaine}.{extension}".format(prot
```

Le principal avantage de la méthode format est que vous n'avez plus besoin d'indiquer l'endroit de votre script et ne l'utiliser que plus tard.

```
# constants.py
```

```
BONJOUR = "Bienvenue {prenom}, vous avez regardé {nombre}
```

```
AU_REVOIR = "À bientôt {prenom} !"
```

```
# main.py
```

```
from constants import BONJOUR
```

```
user = input("Entrez votre nom d'utilisateur : ")
```

```
progression = 2
```

```
message_de_bienvenue = BONJOUR.format(prenom=user, nombre
```

```
print(message_de_bienvenue)
```

```
--->Entrez votre nom d'utilisateur : jugo
```

```
    Bienvenue jugo, vous avez regardé 2 vidéos cette sem
```

raw strings:

les chaînes brutes (raw strings) sont créées en préfixant la chaîne par 'r'. Elles sont utiles lorsque vous voulez inclure des caractères d'échappement dans une chaîne.

3/f-strings :

- permet d'insérer des expressions dans des chaînes de caractères
- Ces expressions servent à insérer des variables dans les chaînes de caractères
- on peut écrire du code dans les accolades
- variable introduit directement dans les chaînes de caractères
- concaténation des chaînes de caractères sans "+"
- formatage simplifié des nombres

- alignement du texte simplifié
- temps d'exécution plus court

```

protocole = "https://"
nom_du_site = "docstring"
extension = "fr"
page = "glossaire"

# Modifiez le code à partir d'ici
URL = f"{protocole}www.{nom_du_site}.{extension}/{page}/"
url = f"{protocole}www.{nom_du_site.lower()}.{extension}"
print(url)
-->https://www.docstring.fr

```

```

liste = ['Pommes', 'Poires', 'Bananes']
print(f"Voici votre liste de courses : {liste}")
---->Voici votre liste de courses : ['Pommes', 'Poires',

```

```

liste = ['Pommes', 'Poires', 'Bananes']
print(f"Voici votre liste de courses : {' , '.join(liste)}")
----->Voici votre liste de courses : Pommes, Poires, Bananes

```

```

liste = ['Pommes', 'Poires', 'Bananes']
liste_final = [fruit.upper() if fruit.lower() == 'pommes' else fruit for fruit in liste]
liste_format = ' , '.join(liste_final)
print(f"Voici votre liste de courses : {liste_format}")
----># Voici votre liste de courses : POMMES, POIRES, BANANES

```

```

>>> prenom = "Pierre"
>>> f"Je me nomme {{prenom}}"
'Je me nomme {prenom}'

```

```

>>> f"Je me nomme {{{prenom}}}"
---->'Je me nomme {Pierre}' En triplant les accolades, c'est la même chaîne
associée à la variable prenom.

```

```

profession = {'Pierre': 'ingénieur'}

```

```

phrase = f"Pierre est {profession['Pierre']}"
phrase = f'Pierre est {profession["Pierre"]}'
print(phrase)
--->Pierre est ingénieur

nombre = 357568.12312
nombre2 = 568.568768
print(f'{nombre : >+20_.4f} {nombre2 : >+20_.4f}')
--->#   +357_568.1231      +568.5688

```

formater une date:

```

from datetime import datetime
f'{datetime(2020, 8, 7, 9, 0):Le %Y-%m-%d à %H:%M}'
# 'Le 2020-08-07 à 09:00'

```

couper un sting en un nombre limite :

```

phrase = """Lorem ipsum dolor sit amet, consectetur adipiscing elit.
            Ut enim ad minim veniam, quis nostrud exercitation ullamco
            Duis aute irure dolor in reprehenderit in voluptate velit
            Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```

```

description = f"{phrase:.60}"
print(description)
# 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut enim ad minim veniam, quis nostrud exercitation ullamco Duis aute irure dolor in reprehenderit in voluptate velit Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.'

```

Comment savoir quelle méthode utiliser ?

La méthode de formatage avec le symbole % est à proscrire depuis plusieurs années.

La méthode format reste encore à privilégier dans certains cas ci-dessus.

Pour tout le reste, les f-string sont vos amis 😊



singleton et Small Integer Caching :

```
{Pour des raisons d'optimisation  
singleton (des objets unique) none true false  
>>> id(True)  
140719121750888  identifiant unique en memoire
```

Python met en cache les entiers compris entre -5 et 256.
lorsque vous créez un entier dans cette plage, vous ne créez pas un nouvel objet.
des raison d'optimisation
C'est ce qu'on appelle la mise en cache de petits entiers.

```
>>> a = 400  
>>> b = 400  
>>> id(a) == id(b)  
False  
ici a et b sont different car on a depasser -5 et 256 de
```

pareil pour les chaines de caractere au dessus de 20 caracteres
ec a = " bonjour jugurta adjoud "
b = " bonjour jugurta adjoud "
>>> id(a) == id(b)
False

```
>>> a = 50  
>>> b = 50  
>>> id(a) == id(b)  
True  
ici a et b sont un meme objet nombre entier 50 car quand on la utiliser dans b sans crer un autre aubjet de type
```

True =1 False= 0
on peut realiser des opeation avec

```
print(True - 1)
print(False + 1)
```



generer un documentation complete avec les docstrings



DocStrings :

Le mot **docstring** est un raccourci pour **DOC**umentation **STR**INGS.

On les utilise pour donner des indications sur le comportement d'une fonction, d'une méthode, d'une classe ou même d'un module entier.

Il y a différents formats que vous pouvez adopter (Google, epytext, rest Numpy, Sphinx, etc..).

l'extension **Python Docstring Generator** énérer des docstring automatiquement selon différents formats. sur visual c s

```
def print_modulo():
    """Affiche tous les chiffres pairs compris entre 0 et 9"""
    for i in range(10):
        if i % 2 == 0:
            print(i)
print_modulo()
print(print_modulo.__doc__) #accéder à cette documentation
0
2
4
6
8
Affiche tous les chiffres pairs compris entre 0 et 1

formatde epytext :
"""docstring epytext
```

```
@param parametre1 : ...
@param parametre2 : ...
@return: description du return
```

```
"""
```

```
formatde google:
```

```
"""docstring google
```

```
Args
```

```
    param1: ...
```

```
    param2: ...
```

```
Returns:
```

```
    description du return
```

```
"""
```

avec l'extension autodocstring de vscode on tape """ et s

```
def fonctionex(nom,age):
```

```
    """description fonction
```

```
    Args:
```

```
        nom (str): nom user
```

```
        age (int): age user
```

```
    Returns:
```

```
        str: nom et age
```

```
    """
```

```
    return nom + str(age)
```

```
print(fonctionex("jugo" , 21) )
```

```
print(fonctionex.__doc__)
```



[les exos de la formation python](#)

1- LA CALCULATRICE SIMPLE :

```
nombre1 = input("saisir un premier nombre :")
nombre2 = input("saisir un deuxieme nombre :")
addision = nombre1 + nombre2
resultat = f"Le résultat de l'addition du nombre {nombre1} + {nombre2} = {addision}"
if not nombre1.isdigit() & nombre2.isdigit():
    print("votre entre n'est pas valide veuillez saisir un nombre valide")
else:
    print(resultat)
```

2-DOCSTRINGS:

```
def print_modulo():
    """Affiche tous les chiffres pairs compris entre 0 et 10"""
    for i in range(10):
        if i % 2 == 0:
            print(i)
print_modulo()
print(print_modulo.__doc__)
```

3-comparaison nmbr aleatoire:

```
import random

a = random.randint(0,50) #génère un nombre aléatoire entre 0 et 50
b = random.randint(0,50)

if a > b :
    print(f"{a} et {b}")
    print("Le nombre a est plus grand que le nombre b.")
elif b > a :
    print(f"{a} et {b}")
    print("Le nombre b est plus grand que le nombre a.")
else :
    print(f"{a} et {b}")
    print("Le nombre a et le nombre b sont égaux.")
```

4- crer un dossier

```

import os
from pprint import pprint

chemin = r"C:\Users\User\Desktop\Udemy\UdemyPython"
dossier = os.path.join(chemin, "dossier", "test")
if not os.path.exists(dossier):
    os.makedirs(dossier)

5- calculatrice :

a=b="" # on declare 2 var a et b de type string vide
while not (a.isdigit() and b.isdigit()):

    a = input("Entrez un premier nombre : ")
    b = input("Entrez un deuxième nombre : ")

    if not (a.isdigit() and b.isdigit()):
        print("Veuillez entrer deux nombres valides")

print(f"Le résultat de l'addition de {a} avec {b} est ég

7- Le nombre mystere :

from random import randint

number_to_find = randint(0, 100)
remaining_attempts = 5

print("*** Le jeu du nombre mystère ***")

# Boucle principale
while remaining_attempts > 0:
    print(f"Il te reste {remaining_attempts} essais{'s' if remaining_attempts > 1 else ''}")

    # Saisie de l'utilisateur
    user_choice = input("Devine le nombre : ")
    if not user_choice.isdigit(): # si la condition = fa

```



```

        print("Veuillez entrer un nombre valide.")
        continue # pour passer le reste du script et rev

user_choice = int(user_choice)

if number_to_find > user_choice: # si cette condition
    print(f"Le nombre mystère est plus grand que {user_choice}")
elif number_to_find < user_choice: # Plus petit
    print(f"Le nombre mystère est plus petit que {user_choice}")
else: # Égal (succès)
    break # pour quitter la boucle

remaining_attempts -= 1

# Gagné ou perdu
if remaining_attempts == 0:
    print(f"Dommage ! Le nombre mystère était {number_to_find}")
else:
    print(f"Bravo ! Le nombre mystère était bien {number_to_find}")
    print(f"Tu as trouvé le nombre en {6 - remaining_attempts} tentatives")

print("Fin du jeu.")

```

8-JEU DE ROLE:

Règles du jeu

Le but de ce projet est de créer un jeu de rôle textuel.

Le jeu comporte deux joueurs : vous et un ennemi.

Vous commencez tous les deux avec 50 points de vie.

Votre personnage dispose de 3 potions qui vous permettent de récupérer un nombre aléatoire de points de vie.

L'ennemi ne dispose d'aucune potion.

Chaque potion vous permet de récupérer un nombre aléatoire de points de vie.

Votre attaque inflige à l'ennemi des dégâts aléatoires c

L'attaque de l'ennemi vous inflige des dégâts aléatoires

Lorsque vous utilisez une potion, vous passez le prochain

À chaque tour, vous attaquez en premier. Il ne peut donc

```
import random
```

```
ENEMY_HEALTH = 50
```

```
PLAYER_HEALTH = 50
```

```
NUMBER_OF_POTIONS = 3
```

```
SKIP_TURN = False
```

```
while True:
```

```
    # Jeu du joueur
```

```
    if SKIP_TURN:
```

```
        print("Vous passez votre tour...")
```

```
        SKIP_TURN = False
```

```
    else:
```

```
        user_choice = ""
```

```
        while user_choice not in ["1", "2"]:
```

```
            user_choice = input("Souhaitez-vous attaquer
```

```
        if user_choice == "1": # Attaque (input retourne
```

```
            #Attaque (quand le user tape 1 c
```

```
            your_attack = random.randint(5, 10)
```

```
            ENEMY_HEALTH -= your_attack
```

```
            print(f"Vous avez infligé {your_attack} poin
```

```
        elif user_choice == "2" and NUMBER_OF_POTIONS >
```

```
            potion_health = random.randint(15, 50)
```

```
            PLAYER_HEALTH += potion_health
```

```
            NUMBER_OF_POTIONS -= 1
```

```
            SKIP_TURN = True
```

```
            print(f"Vous récupérez {potion_health} point
```

```
        else:
```

```
            print("Vous n'avez plus de potions...")
```

```
            continue # il ya plus de potion on ignore le
```

```

if ENEMY_HEALTH <= 0:
    print("Tu as gagné ?")
    break # si l'ennemi n'a plus de point de vie alors

# Attaque de l'ennemi
enemy_attack = random.randint(5, 15)
PLAYER_HEALTH -= enemy_attack
print(f"L'ennemi vous a infligé {enemy_attack} points de dégâts")

if PLAYER_HEALTH <= 0:
    print("Tu as perdu ?")
    break # quitter la boucle

# Stats
print(f"Il vous reste {PLAYER_HEALTH} points de vie.")
print(f"Il reste {ENEMY_HEALTH} points de vie à l'ennemi.")
print("-" * 50)

print("Fin du jeu.")
# Retrouvez la solution de cet exercice dans les fichiers
# https://github.com/DocstringFr/la-formation-complete-py

```

6- gérer une liste de courses : la liste est uniquement

Déroulé du script

Le programme doit permettre de réaliser 5 actions :

- 1- Ajouter un élément à la liste de courses
- 2- Retirer un élément de la liste de courses
- 3- Afficher les éléments de la liste de courses
- 4- Vider la liste de courses

5- Quitter le programme

Tu dois donc demander à l'utilisateur de choisir parmi u

Tu dois gérer le cas de figure où l'utilisateur ne rentre un autre symbole invalide. Dans ce cas, tu dois afficher choisisse une option valide.

```
import sys
```

```
LISTE = []
```

```
MENU = """Choisissez parmi les 5 options suivantes : #cl
1: Ajouter un élément à la liste
2: Retirer un élément de la liste
3: Afficher la liste
4: Vider la liste
5: Quitter
? Votre choix : """
```

```
MENU_CHOICES = ["1", "2", "3", "4", "5"]
```

```
while True: #boucle infinie
    user_choice = input(MENU) # on a transformé la variable en string
    if user_choice not in MENU_CHOICES:
        print("Veuillez choisir une option valide...")
        continue

    if user_choice == "1": # Ajouter un élément
        item = input("Entrez le nom d'un élément à ajouter : ")
        LISTE.append(item)
        print(f"L'élément {item} a bien été ajouté à la liste.")
    elif user_choice == "2": # Retirer un élément
        item = input("Entrez le nom d'un élément à retirer : ")
        if item in LISTE:
            LISTE.remove(item)
```

```

        print(f"L'élément {item} a bien été supprimé")
    else:
        print(f"L'élément {item} n'est pas dans la liste")
    elif user_choice == "3": # Afficher la liste
        if LISTE: # par défaut une liste vide est = False
            print("Voici le contenu de votre liste :")
            for i, item in enumerate(LISTE, 1): #Pour boucler sur la liste
                print(f"{i}. {item}")
        else:
            print("Votre liste ne contient aucun élément")
    elif user_choice == "4": # Vider la liste
        LISTE.clear()
        print("La liste a été vidée de son contenu.")
    elif user_choice == "5": # Quitter
        print("À bientôt !")
        sys.exit()

print("-" * 50)

```

7 -la liste de course avec sauvegard dans le disque dur:

ajouter des lignes de code au fichier de départ afin de sauvegarder la liste dans le même dossier que le script python.

Si c'est le cas, tu dois lire le contenu de ce fichier et l'ajouter à la liste existante. Sinon, on part avec une liste vide. Finalement, quand l'utilisateur choisit de quitter le programme, on sauvegarde la liste sur le disque dans le fichier liste.json.

Pour récupérer le chemin du dossier courant, vous pouvez utiliser la fonction `os.path.dirname(__file__)`

La variable `__file__` est définie automatiquement par Python. N'hésitez pas à visionner de nouveau la partie sur le module `os`.

```

import os
import sys
import json

```

```

CUR_DIR = os.path.dirname(__file__)
LISTE_PATH = os.path.join(CUR_DIR, "liste.json")

MENU = """Choisissez parmi les 5 options suivantes :
1: Ajouter un élément à la liste
2: Retirer un élément de la liste
3: Afficher la liste
4: Vider la liste
5: Quitter
? Votre choix : """

MENU_CHOICES = ["1", "2", "3", "4", "5"]

if os.path.exists(LISTE_PATH): # si le chemin ou il ya r
    with open(LISTE_PATH, "r") as f:
        LISTE = json.load(f)
else:
    LISTE = []

while True:
    user_choice = ""
    while user_choice not in MENU_CHOICES:
        user_choice = input(MENU)
        if user_choice not in MENU_CHOICES:
            print("Veuillez choisir une option valide...")

    if user_choice == "1": # Ajouter un élément
        item = input("Entrez le nom d'un élément à ajout
        LISTE.append(item)
        print(f"L'élément {item} a bien été ajouté à la
    elif user_choice == "2": # Retirer un élément
        item = input("Entrez le nom d'un élément à retir
        if item in LISTE:
            LISTE.remove(item)
            print(f"L'élément {item} a bien été supprimé
        else:

```

```

        print(f"L'élément {item} n'est pas dans la liste")
    elif user_choice == "3": # Afficher la liste
        if LISTE:
            print("Voici le contenu de votre liste :")
            for i, item in enumerate(LISTE, 1):
                print(f"{i}. {item}")
        else:
            print("Votre liste ne contient aucun élément")
    elif user_choice == "4": # Vider la liste
        LISTE.clear()
        print("La liste a été vidée de son contenu.")
    elif user_choice == "5": # Sauvegarder et quitter
        with open(LISTE_PATH, "w") as f:
            json.dump(LISTE, f, indent=4)
        print("À bientôt !")
        sys.exit()

print("-" * 50)

```

8 -le trieur de fichier :

créer un programme pour trier des fichiers selon leur extension. C'est un des premiers programmes que j'ai fait quand j'apprenais Python. Vous pouvez par exemple l'exécuter à intervalle régulier avec un cron pour automatiquement tous les fichiers qui s'y trouvent.

```

"""
Trier les fichiers contenus dans le dossier data selon leur extension.
mp3, wav, flac : Musique
avi, mp4, gif : Videos
bmp, png, jpg : Images
txt, pptx, csv, xls, odp, pages : Documents
autres : Divers
"""

from pathlib import Path
import sys #Ce module fournit un accès à certaines variables d'environnement

EXTENSIONS_MAPPING = {".mp3": "Musique",

```

```

        ".wav": "Musique",
        ".mp4": "Videos",
        ".avi": "Videos",
        ".gif": "Videos",
        ".bmp": "Images",
        ".png": "Images",
        ".jpg": "Images",
        ".txt": "Documents",
        ".pptx": "Documents",
        ".csv": "Documents",
        ".xls": "Documents",
        ".odp": "Documents",
        ".pages": "Documents"}

```

```

BASE_DIR = Path('/Users/thibh/trieur_fichiers/data')
# ou
#sys.argv[-1] #ca permet de recuperer les argument passe
# si lutilisateur met apres ce fichier le nom du dossier
# python ESSAI.py
# ['ESSAI.py']
# python ESSAI.py /User/desktop
# ['ESSAI.py', '/User/desktop']
#ou
# BASE_DIR = input("entrez un chemin de dossier que vous

# On récupère tous les fichiers dans le dossier de base
files = [f for f in BASE_DIR.iterdir() if f.is_file()]
for file in files: # On boucle sur chaque fichier

    # On récupère le dossier cible à partir du dictionnaire
    dossier_cible = EXTENSIONS_MAPPING.get(file.suffix,
    # On concatène le dossier de base avec le dossier ci
    dossier_cible_absolu = BASE_DIR / dossier_cible
    # On crée le dossier cible s'il n'existe pas déjà
    dossier_cible_absolu.mkdir(exist_ok=True) # si le do
    # On concatène le dossier cible avec le nom du fichi
    fichier_cible = dossier_cible_absolu / file.name

```



```
# On déplace le fichier
file.rename(fichier_cible) # renommer le fichier ce c
```

9 - le createur de dossier :

```
"""
```

Il est très courant de devoir créer des structures de do
Quand on ne sait pas programmer, on fait ça à la main, c

crer une structure de dossier avec des sous dossier
cas pratique : debut de projet logiciel on a besoin du c

chaque dossier a 3 sous dossier

```
"""
```

```
from pathlib import Path
```

```
chemin = "/Users/thibh/dossier_test"
```

```
d = {"Films": ["Le seigneur des anneaux",
              "Harry Potter",
              "Moon",
              "Forrest Gump"],
     "Employes": ["Paul",
                  "Pierre",
                  "Marie"],
     "Exercices": ["les_variables",
                   "les_fichiers",
                   "les_boucles"]}
```

```
for dossier_principal, sous_dossiers in d.items(): # la
    for dossier in sous_dossiers:
        chemin_dossier = Path(chemin) / dossier_principal
        chemin_dossier.mkdir(exist_ok=True, parents=True)
```

10 - organiser des donnees :

```

with open("/Users/thibh/Documents/prenoms.txt", "r") as
    lines = f.read().splitlines()

prenoms = []
for line in lines:
    prenoms.extend(line.split())

prenoms_final = [prenom.strip(",. ") for prenom in prenoms]

with open("/Users/thibh/Documents/prenoms_final.txt", "w") as f:
    f.write("\n".join(sorted(prenoms_final)))

```

Voici un exemple simple qui copie tous les fichiers d'un répertoire source vers un répertoire de destination.

```

import shutil
import os

source_directory = "/chemin/vers/source"
destination_directory = "/chemin/vers/destination"

# Liste de tous les fichiers dans le répertoire source
files = os.listdir(source_directory)

# Copie de chaque fichier vers le répertoire de destination
for file in files:
    source_path = os.path.join(source_directory, file)
    destination_path = os.path.join(destination_directory, file)
    shutil.copy(source_path, destination_path)

```

Cet exemple simple illustre comment Python peut être utilisé pour copier des fichiers d'un répertoire à un autre.



Les structures conditionnelles

IF :

```
if condition:
    code
    ...
| Python exécute le code qui se trouve à l'intérieur de
condition est vraie (True).

| Si c'est False, le code n'est pas exécuté.

| En Python, None et 0 sont des valeurs interprétées com
```

```
age = 18
MAJORITE = 18

if age > MAJORITE:
    print("Je suis majeur !")
elif age == MAJORITE:
    print("Tout juste majeur depuis aujourd'hui")
else:
    print("Je suis mineur..")

print("J'ai", age, "ans !")
```

```
| L'ordre des structures conditionnelles a une importance
| Différentes structures conditionnelles peuvent retourner
| Avec une même suite de if / elif, Python n'exécutera qu'une seule
Dès qu'une condition est satisfaite, Python exécutera le code correspondant
et sortira de la structure conditionnelle.
```

```
age = 24
MAJORITE = 18

if age >= MAJORITE:
    print("Je suis majeur !")
```

```
print("J'ai", age, "ans !")
#Si la variable age est égale à 18 ou plus, j'indente mo
#Dans tous les cas, le programme se termine par l'affich
```

Imbriquer des structures conditionnelles

```
age = 18
MAJORITE = 18

if age >= MAJORITE:
    print("Je suis majeur !")
    if age == MAJORITE:
        print("Tout juste majeur depuis aujourd'hui")
else:
    print("Je suis mineur..").
```



LES BOUCLES : repeter une operation un certain nombre de fois , parcourir des structure de donner(lists , string , dictionnaire ,tuples..) des iterables ex dutilisation : passer sur une liste de fichier pour les trier et supprimer les fichier les plus volumineux

La boucle FOR :

```
for value in object:
    code
    ....
```

```
for i in [1,2,3,4,5]: # }5 itterations python va ce char
```

```
for i in range(500): boucler sur une liste de nombre de
```

```

liste = ["python","javascript","php"]
for langage in liste:
    if langage[0]== "p":
        print(langage)---> python php

prenom = "John"
for lettre in prenom:
    print(lettre)

for (nbr, name) in [(1, "Patrick"), (2, "John"), (3, "Ma
    print(nbr, name)

enumerate: la fonction native enumerate() permet de parc
un objet de type enumerate qui contient l'indice et l'él
l'utiliser

enumerate(iterable, start=0)
iterable : un objet qui supporte l'itération
start : le chiffre par lequel on commence l'énumération,

ex :
l1 = ["eat", "sleep", "repeat"]
for count, ele in enumerate(l1, 100):
    print (count, ele)
-->100 eat
    101 sleep
    102 repeat

lettres = ['a', 'b', 'c']
for position, lettre in enumerate(lettres):
    print(position, lettre)
-->0 a
    1 b
    2 c

ma_liste = ['comment', 'coder', '.com']
for position, mot in enumerate(ma_liste):

```

```

    print(f'{position} : {mot}')
-->0 : comment
    1 : coder
2 : .com

```

FOR ELSE :

```

liste = ["python","javascript","php"]
for lng in liste:
    print(lng)
else:
    print("vide")

```

```

liste = ["python","javascript","php"]
for lng in liste:
    if lng == "php":
        print("php est dans la liste")
        break
else:
    print("php introuvable")

```

boucle WHILE : executer tanque une condition est vrai --
 Lorsque votre script rencontre une boucle while, il véri

```

i = 0
while i < 500:
    print("bonjour")
    i += 1 # si on enleve cette incrementation alros la

```

```

continuer = "o"
while continuer == "o":
    print("on continue")
    continuer = input("vous les vous continuer ? : ")

```

```

import time
while True: # while true permet de répéter indéfiniment

```

```
print("sauvegarde en cours..")
time.sleep(600) # attendre 600 seconde entre chaque
```

On déclare la variable qui nous sert de condition avant
⇒ On assigne la valeur 0 à la variable i juste avant d'

On met à jour notre condition à chaque nouvelle itération
⇒ On incrémente i à la fin de ma boucle

Ce faisant, on évite de rentrer dans une boucle infinie.

```
liste = []
while liste == True: #une liste vide est False. Si la li
    print("Itération sur la liste") # il sera pas execut
liste = ["python","javascript","php"]
for lng in liste:
    if lng.isdigit():
        continue
    print(lng)
```

WHILE ELSE :

```
i = 0
while i < 3:
    print('Salut')
    i += 1
else:
    print('Au revoir')
```

contrôler une boucle avec continue et break :

1-continue

```
liste = ["1","4","python","2","javascript","php"]
for i in liste:
    if i.isdigit():
        continue # si i est un chiffre alors continuer
```

```

# continue n'arrête pas l'exécution
# elle permet juste de passer à l'itération suivante
# donc quand i est divisible par 2
print(i) # python javascript php

```

2-break

```

fruits = ['🍊', '🍋', '🍏', '🍒', '🍑']
fruits_manges = 0

for fruit in fruits:
    print("Je mange des " + fruit)
    fruits_manges += 1
    if fruits_manges == 3:
        break # break permet d'interrompre l'exécution d'un bloc

print("Je n'ai plus faim !") # Je mange des 🍊

```

Je mange des 🍏
Je n'ai plus faim !

Dans le cas d'un bloc d'instruction à l'intérieur d'un autre, l'instruction break ne sortira que du bloc d'instruction interne.

```

for a in range(5):
    for b in range(5):
        print("a:", a, "b: ", b)
        if b == 3:
            break

```

liste comprehension : permet d'iterer sur une liste et de créer une nouvelle liste à partir de celle-ci.

```

EX: afficher que les nombres positifs
liste = [-5, 2, 3, 4, -2, -7, -6]
nombres_positifs = []
for i in liste:
    if i > 0 :
        nombres_positifs.append(i)

```



```

print(nmbr_positif) # [2, 3, 4]

liste = [-5,2,3,4,-2,-7-6]
nmbr_positif = [i for i in liste if i > 0] --> comprehension
print(nmbr_positif) # [2, 3, 4]
-----

nombres = [1, 21, 5, 44, 4, 9, 5, 83, 29, 31, 25, 38]
nombres_pairs = []
for i in nombres:
    if i % 2 == 0:
        nombres_pairs.append(i)

print(nombres_pairs)

nombres = [1, 21, 5, 44, 4, 9, 5, 83, 29, 31, 25, 38]
nombres_pairs = [i for i in nombres if i % 2 == 0] --> comprehension
print(nombres_pairs)
-----

nombres = range(10)
nombres_inverses = []
for i in nombres:
    if i % 2 == 0:
        nombres_inverses.append(i)
    else:
        nombres_inverses.append(-i)

print(nombres_inverses)

nombres = range(10)
nombres_inverses = [i if i % 2 == 0 else -i for i in nombres]
print(nombres_inverses)
-----

nombres = range(51)
nombres_pairs = []
for i in nombres:

```

```

    if i % 2 == 0:
        nombres_pairs.append(i)

nombres = range(51)
nombres_pairs = [i for i in nombres if i % 2 == 0 ] -->
print(nombres_pairs)

fonction Any ET All : operent sur les listes pour verif
                        ou si toutes

ex = any([True,False,False]) --> il suffit qu'une valeur
print(ex) --> True

notes = [12,11,15,18,8]
if any([x >= 18 for x in notes]): # si il ya aumoin une
    print("excelent")

ex = all([True,False,False]) --> il faut que toute les v
print(ex) --> False

files = ["jugo.jpg", "snake.py", "photo.jpg"]
if all([f.endswith(".jpg") for f in files]): # verifier
    print("..")

exos sur les boucles :
1-
chaine = "Python"
for i in reversed(chaine):
    print(i) # n o h t y p
ou :
mot = "Python"
for lettre in mot[::-1]:
    print(lettre)
2-
for i in range(1, 11):
    print(f"utilisateur {i}") # utilisateur 1 .....2 .....

```

3-

```
mot = "Python"
for i in range(len(mot)): # range a besoin d'un nombre pour la fin
    print(i) # d'afficher l'index de chaque lettre du mot
```

4-

```
nombre = 7
for i in range(11):
    print(f" {i} x {nombre} = {i*nombre}") # afficher la table de 7
```

5-

```
continuer = "o"
while continuer == "o":
    print("On continue !")
    result = input("Voulez-vous continuer ? o/n ")
    if result == "o":
        continue
    elif result == "n":
        break
    else:
        print("taper o ou n")
        continuer
```



opérateur logique

AND

il suffit qu'un seul des opérandes soit faux pour que l'expression soit fautive

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

L'opérateur and a précedence sur les opérateurs or et not

Il faut donc faire attention dans le cas où plusieurs ex importance.

Il peut être nécessaire d'utiliser des parenthèses afin opérateurs.

OR

il suffit qu'un seul des deux opérandes soit vrai pour c

```
True or True = True
```

```
True or False = True
```

```
False or True = True
```

```
False or False = False
```

```
prenom = input("Entrez votre prénom : ") or "Anonyme"
print(prenom)
```

NOT

```
if not (nombre1.isdigit() and nombre2.isdigit()):
    print("votre entre n'est pas valide veuillez saisir u
```



modules et fonction : un module est un fichier python qu'on doit importer qui contient des fonctions déjà écrites qu'on peut utiliser on les appelle aussi bibliothèques ou librairies ex pour le module random c'est un fichier python random.py qui à l'intérieur a des fonctions comme randint() *

on peut nous même créer des modules qu'on utilisera dans d'autres fichiers python * À l'intérieur d'un

module, son propre nom est accessible par la variable `__name__`

Les développeurs de Python ont mis au point de nombreux modules qui effectuent une quantité phénoménale de tâches. Pour cette raison, prenez toujours le réflexe de vérifier si une partie du code que vous souhaitez écrire n'existe pas déjà sous forme de module. syntaxe

conseillée —> **module.fonction()**

module random generer des nombres aleatoire

ex

```
>>> import random
```

```
>>> random.randint(0, 10)# Cette fonction renvoie un non  
# 10 inclus.
```

```
4
```

```
>>> random.uniform(0, 10)# Cette fonction renvoie un non  
#entre 0 inclus et 10 inclus.
```

```
5,6394039
```

```
>>> random.randrange(900)# on lui donne un seul argument  
#aléatoirement entre 0 inclus et 900 exclu.
```

```
>>> random.randrange(0, 900, 10) # renvoie un nombre ent  
#exclu avec un pas de 10
```

module math

```
>>> import math
```

```
>>> math.cos(math.pi / 2)
```

autres syntaxes ou facons d'utiliser les modules

avec from, on peut importer une fonction spécifique d'un module
en question est requis

```
>>> from random import randint
```

```
>>> randint(0,10)
```

```
7
```

```
from random import *
```

importe toutes les fonctions du module random.

On peut ainsi utiliser toutes ses fonctions directement,

Il est également possible de définir un alias (un nom plus court)

```
>>> import random as rand
```

```
>>> rand.randint(1, 10)
```

```
6
```

pour vider de la mémoire un module déjà chargé, on peut utiliser del

```
pour connaître d'un seul coup d'œil toutes les méthodes
fonction dir()
>>> import random
>>> print(dir(random))
```

```
from pprint import pprint
pprint(dir(random))
```

```
voir l'aide sur une fonction d'un module
help(random.randint)
```

```
>>> pprint(dir(os.path))
help(os.path)
quelque modules frequents
```

math : fonctions et constantes mathématiques de base (si
sys : interaction avec l'interpréteur Python, passage d'
os : dialogue avec le système d'exploitation (cf. plus l
pathlib : Ce module offre des classes représentant le sy
pour différents systèmes d'exploitation
random : génération de nombres aléatoires.
time : accès à l'heure de l'ordinateur et aux fonctions
urllib : récupération de données sur internet depuis Pyt
Tkinter : interface python avec Tk. Création d'objets gr
et Tkinter).
re : gestion des expressions régulières (cf. chapitre 16

* creer un module

ex: créez un fichier fibo.py dans le répertoire courant
qui contient

```
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

```

on ouvre l'interpreteur py
>>> import fibo
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

```

```

module OS : gerer les fichier du systeme d'exploitation

import os

# creer un dossier
chemin = r"C:\Users\User\Desktop\Udemy\UdemyPython" #le
# raw string Python traite le caractère barre oblique

os.path – manipulation courante des chemins

dossier = os.path.join(chemin, "dossier", "test") # on \
#au chemin avec la fonction join car elle gere les slash
#concatenation avec f string mais la methode join est plus simple
os.makedirs(dossier) # on creer le dossier makedirs permet de creer un dossier
#contrairement a mkdir

pour ne pas avoir d'erreur quand on supprime un dossier
os.removedirs(dossier) # removedirs supprime tout les dossier dans le dossier
#courant
if os.path.exists(dossier):
    os.removedirs(dossier)

pour ne pas avoir d'erreur quand on creer un dossier deja existant
if not os.path.exists(dossier): # os.path.exists True si le dossier existe
#descripteur de fichier ouvert.
    os.makedirs(dossier) # si le chemin specifie en parametre existe
#alors on le cree
else:
    print("dossier existe deja")

ou os.makedirs(dossier, exist_ok=True) marche que sur python 3.2 et plus
callable() voir si une fonction est appellable

```

```
ex
print(callable(os.path))
```



trier une liste

```
words = ['D', 'A', 'C', 'B']

words.sort()
# word.sort(reverse=True) ['D', 'C', 'B', 'A'] modifie la liste
# .sorted() obtenir une nouvelle copie triée d'une liste
print(words)          # ['A', 'B', 'C', 'D']
```

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with
>>> letters[:] = []
>>> letters
[]
```

les methodes se sont comme des fonction maisqui sont ass

append(x) Ajoute un élément à la fin de la liste. Équiva
Étend la liste en y ajoutant tous les éléments de l'itér

Supposons que vous ayez a = [0,1,2] et que vous fassiez

ce que vous dites à Python, c'est que juste après [0,1,2] il devrait y en avoir 3,4,5. Ainsi ça deviendra [0,1,2,3,4,5] sera effectivement égal à [3,4,5] comme vous l'avez déclaré.



Les Listes : Les listes font parties des quatre grandes **structures de données** qui existent en Python en plus des **sets**, des **tuples** et des **dictionnaires**. une structure de données **muable (modifiable)** et **ordonnée** dans laquelle tu peux stocker **n'importe quel type d'objet**.(élément)

On accède aux éléments d'une liste grâce à leur **indice**, c'est-à-dire leur position dans la liste.

une liste est une **séquence de données** ! Cela signifie que tu peux **itérer** sur cette séquence, avec une boucle **for** par exemple.

Une liste est ordonnée selon la façon dans laquelle nous les ordonnons

```
# liste vide
villes = []

# liste avec trois éléments de type str
villes = ['Paris', 'Lille', 'Lyon']

# liste avec cinq éléments de type int
prix = [3, 10, 25, 40, 100]

# liste avec plusieurs éléments de types différents
liste_de_tout_et_rien = [5, 'Docstring', True, 9.5, 4, '']

# liste qui contient un dictionnaire
adresse = [
    {
        'rue': 'rue du Serpent',
        'numero': 6,
        'ville': 'Lille'
    }
]
```

liste() convertire une variable en liste

ex:

```
text = 'Python'
# convert string to list
text_list = list(text)
print(text_list)
# check type of text_list
print(type(text_list)) # Output: ['P', 'y', 't',
```

Dans une liste chaque élément est accessible grâce à sa

```
villes = ['Paris', 'Lille', 'Lyon']
```

```
print(villes[0]) # 'Paris'
```

```
print(villes[1]) # 'Lille'
```

muable (modifiable)

```
villes = ['Paris', 'Lile', 'Lyon']
```

```
villes[1] = 'Lille'
```

```
print(villes) # ['Paris', 'Lille', 'Lyon']
```

Et si on ajoutait une ville ?

```
villes.append('Strasbourg')
```

```
print(villes) # ['Paris', 'Lille', 'Lyon', 'Strasbourg']
```

J'ai jamais aimé Paris...

```
del villes[0]
```

```
print(villes) # ['Lille', 'Lyon', 'Strasbourg']
```

iterable

```
villes = ['Lille', 'Lyon', 'Strasbourg']
```

```
for ville in villes:
```

```
    print(ville.upper())
```

Créer une liste

Avec les crochets

```
liste_vide = []
```

Avec le constructeur

```
liste_vide = list()
```

```
site = 'Docstring'
```

```
site_to_liste = list(site)
```

```
print(site_to_liste) # ['D', 'o', 'c', 's', 't', 'r', 'i', 'n', 'g']
```

Accéder à des éléments

liste[indice] → Retourne l'élément associé à l'indice

liste[début:fin:pas] → Retourne le ou les éléments en fonction

```
          [  0          1          2  ]
villes = ['Paris', 'Lille', 'Lyon']
          [  -3          -2          -1  ]
```

```
print(villes[1]) # 'Lille'
```

```
print(villes[-1]) # 'Lyon'
```

```
j = [1, 2, [1, "python",3], 3]
```

```
print(j[2][1])
```

```
---->python
```

[début(inclu):fin(exclu):pas] #l'indice de début est inclus

```
villes = ['Paris', 'Lille', 'Lyon']
```

```
print(villes[0:1]) # ['Paris'] on commence de 0 on s'arrête à 1
```

```
print(villes[0:2]) # # ['Paris', 'Lille'] /on commence de 0 on s'arrête à 2
```

```
print(villes[0::2]) # ['Paris', 'Lyon'] / on récupère les éléments à l'indice 0, 2, 4, ...
```

```
# jusqu'à la fin
```

```
villes = ['Paris', 'Lille', 'Lyon', 'Lyon', 'Lyon']
```

```
villes = villes[0::4]
```

```

        print(villes)
        ---->['Paris', 'tizi']
print(villes[0:])      # ['Paris', 'Lille', 'Lyon']
print(villes[:])       # ['Paris', 'Lille', 'Lyon']
print(villes[:-1])     # ['Paris', 'Lille']
print(villes[::-1])    # ['Lyon', 'Lille', 'Paris'] / on
                        # jusqu'a la fin

les slices :
word = 'Python'
word[0:2]  # caractères commençant de la position 0 (inclu)
-->'Py'
word[2:5]  # caractères commençant de la position 2 (inclu)
-->'tho'
word[:2]   # caractères commençant du début à la position 2 (exclu)
-->'Py'
word[4:]   # caractères commençant de la position 4 (inclu) jusqu'à la fin
-->'on'
word[-2:]  # caractères commençant de l'avant-dernier (inclu) jusqu'à la fin
-->'on'
word[::2]  # caractères du début à la fin avec pas de 2
-->'pto'
word[1:4:] # caractères de la 1ère position à la 4ème avec pas de 2
print(word)
-->'yth'
word[1:-2:2] # caractères de la 1ère position à l'avant-dernier (exclu) avec pas de 2
print(word)
-->'yh'
word[::-1]  # on inverse l'ordre de la liste (on récupère la liste inversée)
                        # jusqu'a la fin

print(word)
-->'nohtyp'
word[1:-1] # on exclut le premier et le dernier élément
-->'ytho'
word[:2] + word[2:]
-->'Python'
word[:4] + word[4:]
-->'Python'

```

```

liste = ["Maxime", "Martine", "Christopher", "Carlos", 'Eric']

trois_premiers = liste[0:2]
trois_derniers = liste[-3:]
milieu = liste[1:-1]
premier_et_dernier = liste[::5] #du premier au dernier a

```

quelques methodes pour les liste :

```

liste = ["Maxime", "Martine", "Christopher", "Carlos", 'Eric']

```

```

**print(liste.index("Martine")) --->1 #la position d'un element

```

```

**print(liste.count("Martine")) --->1 #nombre d'occurences

```

```

**liste.sort() # La méthode ne retourne rien (None), donc
#vous écrasez votre liste par la valeur None. La méthode
# notre liste, pas besoin donc de faire une assignation.

```

```

print(liste)
-->['Adjoud', 'Carlos', 'Christopher', 'Eric', 'Martine']

```

mais on a la fonction sorted qui elle retourne une liste

```

**liste-trie = sorted(liste)
print(liste-trie)
--->['Adjoud', 'Carlos', 'Christopher', 'Eric', 'Martine']

```

```

**liste.reverse()# elle ne rtourne rien elle va juste di
# on peut pas faire liste = liste.reverse() car sa va r
--->['Eric', 'Adjoud', 'Carlos', 'Christopher', 'Martine']

```

methodes pour enlever des elements d'une liste :

```

liste = ["Maxime", "Martine", "Christopher", "Carlos", 'Eric']
liste.pop(-1) # enlever un element grace a son index
print(liste)

```

```
--->['Maxime', 'Martine', 'Christopher', 'Carlos', 'Adjc  
liste.clear() # supprime tout les elements de la liste
```

Ajouter des éléments

`.append(item)` → Ajoute une seul valeur à la fin de t

```
villes = ['Paris', 'Lille', 'Lyon']  
villes.append('Strasbourg') #['Paris', 'Lille', 'Lyon',
```

`.insert(index, item)` → Ajoute un item à la position

```
villes = ['Paris', 'Lille', 'Lyon']  
villes.insert(1, 'Strasbourg')  
print(villes) # ['Paris', 'Strasbourg', 'Lille', 'Lyon']
```

`.extend(iterable)` → Ajouter plusieurs valeur passer s
à la suite car extend accepte un argument dans ta liste

```
villes = ['Paris', 'Lille', 'Lyon']  
autres_villes = ['Strasbourg', 'Marseille']  
villes.extend(autres_villes)  
print(villes) # ['Paris', 'Lille', 'Lyon', 'Strasbourg']
```

modifier les éléments

`liste[indice] = valeur` → Assigne une nouvelle valeur

```
villes = ['Paris', 'Lille', 'Lyon']  
villes[1] = 'Strasbourg'  
print(villes) # ['Paris', 'Strasbourg', 'Lyon']
```

`liste[début:fin] = [valeur1, valeur2, ...]` → Assigne c

```
villes = ['Paris', 'Lille', 'Lyon']  
villes[1:] = ['Strasbourg', 'Rennes']  
print(villes) # ['Paris', 'Strasbourg', 'Rennes']
```

```
# Changez la position de l'élément 'Python' dans la liste
# à la fin de la liste (["Java", "C++", "Python"])
liste = ["Java", "Python", "C++"]
liste.remove("Python")
liste.append("Python")
```

supprimer des elements

`del liste[indice]` → Supprime un ou plusieurs éléments

```
villes = ['Paris', 'Lille', 'Lyon']
```

```
del villes[0]
print(villes) # ['Lille', 'Lyon']
```

```
villes.clear()-->[]
```

`liste.pop(indice)` → Retire un élément de la liste. S

```
villes = ['Paris', 'Lille', 'Lyon']
```

```
ma_ville = villes.pop(1)
print(villes) # ['Paris', 'Lyon']
print(ma_ville) # Lille
```

`liste.remove(item)` → Supprime un élément de la liste

```
villes = ['Paris', 'Lille', 'Lyon']
```

```
villes.remove('Paris')
print(villes) # ['Lille', 'Lyon']
```

supprimer toute les occurrences d'un element dans une liste

```
#En utilisant remove
def remove_item(lst, item):
```

```

        counter_item = lst.count(item)
        for _ in range(counter_item): # on a utiliser _ car
#la boucle est exécutée jusqu'à présent, mais simplement
#nombre de fois au total.
            lst.remove(item)
        return lst

```

```

lst = [1, 2, 3, 2, 2, 3, 4, 2, 4, 1, 2]
print(remove_item(lst, 2)) ---->[1, 3, 3, 4, 4, 1]

```

#Ou en filtrant

```

def filtered_list(lst, item):
    return [i for i in lst if i != item]

```

```

lst = [1, 2, 3, 2, 2, 3, 4, 2, 4, 1, 2]
print(filtered_list(lst, 2)) ---->[1, 3, 3, 4, 4, 1]

```

joindre les elements d'une liste :

```

methode join()
villes = ['Paris', 'Lille', 'Lyon']
result = (" - ").join(villes) #on ajoute un tiret et 2 e
----> Paris - Lille - Lyon

```

```

for i, element in enumerate(donnees, 1):
    print(f"{i}-{(' ').join(element)}")
["poirre", "ammande"]
1-poirre
2-ammande

```

creer une liste a partir d'une chaine de caractere :

```

villes = "Paris, Lille, Lyon"
result = villes.split()
print(result) --->['Paris,', 'Lille,', 'Lyon']
result = villes.split(", ") # la virgule et lescpace ne
print(result) --->['Paris', 'Lille', 'Lyon']

```



```
#et si on fait un split dun element qui ne se trouve pas
#on aura une liste avec un seul element (toute la chaine)
villes = "Paris, Lille, Lyon"
result = villes.split(",")
print(result)-->['Paris, Lille, Lyon']
```

opérateur d'appartenance : (in , not in) :

```
villes = ['Paris', 'Lille', 'Lyon']
```

```
"Paris" in villes ---> True
```

```
"Paris" not in villes --->False
```

```
users = ['jugo', 'mayas', 'khelif']
```

```
if "jugo" in users:
```

```
    print("bnjr jugo")
```

```
if "jugo" in users:
```

```
    users.remove("jugo")
```

```
    print(users)--->['mayas', 'khelif']
```

```
"java" in "javascript"
```

```
--->True
```

liste Imbriquée:

```
liste = ["python", "javascript", [1, "java",0], 3]
```

```
print(liste[2][1])
```

```
---->java
```

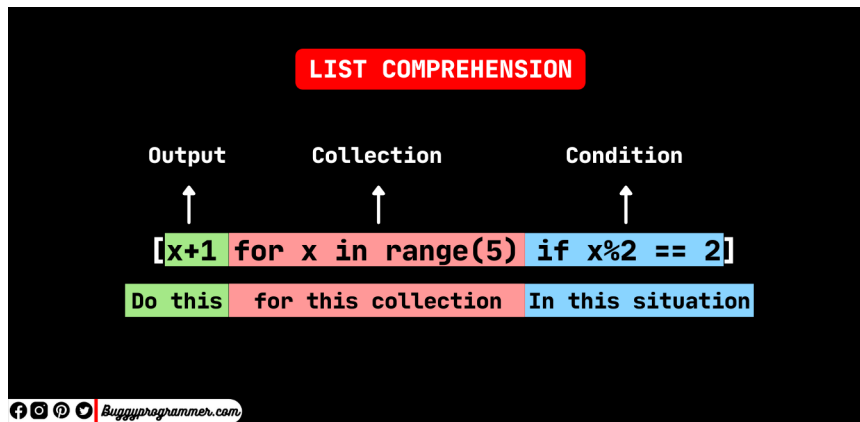
```
print(liste[0][0:2])
```

```
---->py
```

```
nombres = [1, [4, [2, 3]], 5, [6], [[7]]]
```

```
sept = nombres[-1][0][0]--->7
```

Si la liste est vide, elle correspond à la valeur false



liste comprehension:

La compréhension de liste est une syntaxe concise et expressive pour créer des listes en une seule ligne, de manière plus lisible et compacte.

`[nouvelle_valeur for element in sequence if condition]`

- nouvelle_valeur : Expression qui définit la valeur de chaque élément de la nouvelle liste.
- element : Variable qui prend la valeur de chaque élément de la séquence.
- sequence : Séquence (liste, tuple, chaîne de caractères, etc.).
- condition : (Optionnel) Condition permettant de filtrer les éléments.

Exemple Basique :

```
# Création d'une liste des carrés des nombres de 0 à 4
carrés = [x**2 for x in range(5)]
```

Résultat : `[0, 1, 4, 9, 16]`

Exemple avec Condition :

```
# Création d'une liste des carrés des nombres pairs de 0 à 9
carrés_pairs = [x**2 for x in range(10) if x % 2 == 0]
```

Résultat : `[0, 4, 16, 36, 64]`

Exemple avec Transformation :

```
# Création d'une liste des chaînes en majuscules à partir de mots
mots = ['python', 'est', 'cool']
```

```
majuscules = [mot.upper() for mot in mots]
```

```
# Résultat : ['PYTHON', 'EST', 'COOL']
```

Exemple avec Tuple :

```
# Création d'une liste de tuples (nombre, carré) pour les
```

```
tuples_carres = [(x, x**2) for x in range(5)]
```

```
# Résultat : [(0, 0), (1, 1), (2, 4), (3, 9), (4, 16)]
```

La compréhension de liste est une technique puissante et lisible et plus concis, surtout lors de la création de l

les tuples :

est un objet immuable

Les tuples, c'est quasiment la même chose que les listes c'est qu'on ne peut ni ajouter ni enlever d'éléments à u

Les tuples, étant plus restreints en termes de fonctionn de place dans la mémoire de votre ordinateur.

Si vous savez que la taille de vos listes auront un nombr des millions de listes, il peut être préférable de passe

la syntaxe est similaire aux listes sauf qu'on utilise l

```
mon_tuple = (1, 2, 3)
```

Toutes les méthodes pour ajouter et enlever des éléments

```
t = () # Un tuple python vide
```

```
print(t) # ()
```

```
t = ('spam', 'eggs') # Un tuple python contenant deux ch
```

```
print(t) # ('spam', 'eggs')
```

```
t = ('spam', 'eggs', 9, ['one', 'two', 'three'], True) #
```

```
print(t) # ('spam', 'eggs', 9, ['one', 'two', 'three'],
```

```
t = ('spam', 'eggs', (1, 2, 3), 9.4, True) # Un tuple py
```

```
print(t) # ('spam', 'eggs', (1, 2, 3), 9.4, True)
```

il est possible de convertir un tuple en liste et vice-\

```
>>> mon_tuple = (1, 2, 3)
>>> liste = list(mon_tuple)
[1, 2, 3]
>>> mon_tuple = tuple(liste)
(1, 2, 3)

# Ceci n'est pas un tuple
pas_un_tuple = (1)
print(type(pas_un_tuple)) # Résultat : <class 'int'>
cest la virgule qui differenci un tuple dun aautre objet
# Ceci est un tuple avec un seul élément
mon_tuple = (1,)
```

quand utiliser les listes , tuples , dict, set

- Est-ce que j'ai besoin d'associer une valeur à une clé
- Est-ce que j'ai besoin de modifier ma structure de donn
- Est-ce que j'ai besoin de garder les éléments dans le m
- Pour toutes les autres possibilités → Liste

c'est assez simple de passer d'une structure à l'autre g
fonction set() pour convertir ma liste villes d'origir
ce qui va avoir pour effet de retirer tous les doublor

les ensembles set():

Set est une collection non ordonnée de types de données
en double. Le principal avantage de l'utilisation d'un
est qu'il dispose d'une méthode hautement optimisée pour
contenu dans l'ensemble
Les ensembles sont utiles pour effectuer des opérations
la différence, etc

Vous pouvez créer un ensemble en utilisant des accolades
ensemble1 = {1, 2, 3, 4, 5}
ensemble2 = set([3, 4, 5, 6, 7])

Ajout d'éléments :

```
ensemble1.add(6)
```

Suppression d'éléments :

```
ensemble1.remove(3)
```

Vérification d'appartenance :

```
print(2 in ensemble1) # Affiche True
```

Opérations ensemblistes :

Union :

```
ensemble_union = ensemble1.union(ensemble2)
```

ou

```
ensemble_union = ensemble1 | ensemble2
```

Intersection :

```
ensemble_intersection = ensemble1.intersection(ensemble2)
```

ou

```
ensemble_intersection = ensemble1 & ensemble2
```

Copie d'ensemble :

```
copie_ensemble = ensemble1.copy()
```

Vider un ensemble :

```
ensemble1.clear()
```

éliminer les doublons:

```
villes = ['Paris', 'Lille', 'Lyon', 'Paris', 'Strasbourg']
```

```
villes = list(set(villes))
```

```
print(villes) # ['Lyon', 'Lille', 'Strasbourg', 'Paris']
```

```

#C'est une technique courante en Python pour éliminer les doublons
# en ensemble (ce qui supprime les doublons) puis en la transformer en ensemble

mon_ensemble = {10, 20, 30, 40}

String = 'GeeksForGeeks'
set1 = set(String)
print("\nSet with the use of an Object: " )
print(set1)
--->Set with the use of an Object:
      {'G', 's', 'e', 'o', 'r', 'F', 'k'}

# the use of a List # transformer une liste en set
set1 = set(["Geeks", "For", "Geeks"])
print("\nSet with the use of List: ")
print(set1)
----->Set with the use of List:
        {'Geeks', 'For'}

```

METHODE ET FONCTION: déjà definie :

Une méthode est une fonction attachée a une classe. Une fois qu'on peut appeler depuis une autre, avec des entrées et des paramètres. Contrairement aux fonctions les méthodes agissent directement sur les objets. Donc quand on utilise une fonction sur une liste, elle crée une nouvelle liste. Contrairement à une méthode qui modifie directement l'objet.

ex :

```

liste = [5,3,2,6,4,9]
sorted(liste)
print(liste)-->[5, 3, 2, 6, 4, 9]
#la fonction sorted va nous retourner une liste ordonnée
#il faudra écraser la liste d'origine
liste = sorted(liste)-->[2,3,4,5,6,9]

#avec la methode sort() vu qu'une methode est directement

```

```
#la methode va agir directement sur la liste sans faire
liste = [5,3,2,6,4,9]
liste.sort()
```

ex 2:

```
def fonction_int():
    a = "5"
    int(a)
    print(type(a)) --><class 'str'># ca modifie pas la var a
    ---
    a = "5"
    a = int(a)
    print(type(a))--><class 'int'>
```

LES OBJETS MUABLE ET IMMuable :

muables : dictionnaires, liste, sets

ex :

```
liste.append(5) # modifie directement la liste
```

immuable : strings , nombres..

ex :

```
"jugo".title() # ne modifie pas directement la chaine de caractere
titre ="jugo".title() # il faut laffecter a une variable
print(titre)
```

fonction supplémentaire :

len utiliser avec des iterables

```
**len("jugo") --> 4
```

```
len([1, 2, 3]) --> 3 nmbr d'elements dans la liste
```

```
len([1, 2, [3, 4]])-->3
```

```
len({'un': 1, 'deux': 2, 'trois': 3})-->3 retourne le nombre de valeurs
```

Pour permettre à la fonction len d'être utilisée avec des objets personnalisés, il faut implémenter la méthode spéciale `__len__`.

ex : une classe panier

```
class Panier:
```

```

def __init__(self):
    self.articles = []

def ajouter_article(self, article):
    self.articles.append(article)

def __len__(self):
    return len(self.articles)

```

Maintenant, vous pouvez utiliser la fonction len avec de

```

panier = Panier()
panier.ajouter_article("pomme")
panier.ajouter_article("banane")

print(len(panier))  # Résultat: 2

```

(Imaginons que nous ayons une liste de tâches à accomplir pour afficher le pourcentage d'avancement basé sur le r

```

taches = [
    {"nom": "Faire les courses", "termine": True},
    {"nom": "Nettoyer la cuisine", "termine": False},
    {"nom": "Arroser les plantes", "termine": True},
    {"nom": "Appeler le plombier", "termine": False},
]

def pourcentage_avancement(taches):
    termine = [tache for tache in taches if tache["termine"]]
    return (len(termine) / len(taches)) * 100

print(pourcentage_avancement(taches))  # Résultat: 50.0

```

Pass en Python est un espace réservé qui empêche le prog


```

**round(2.3)--> 2 , round(2.7)-->3 arrondir un nmbr deci
**min([1, 2, 3])-->1 , min([a, b, c]) -->a en fonction c
**max([1 ,2 ,3])-->3
**sum([1,2,3])-->6 somme des element dune liste que des

**range(5) // cree une liste de nombres allant de 0 a 5-
    range(2, 5) // crre une liste de nombre allant de 2 a

a = range(5)
print(a)  range(0, 5)
b = list(a)
print(b)  [0, 1, 2, 3, 4]
print(a.start) 0
print(a.step) 1
print(a.stop) 5

for i in range(1, 11):
    print(f"utilisateur {i}") utilisateur 1

mdp = input("saisir un mdp d'aumoin 8 caracteres : ")
mdp_trop_court = "mdp_trop_court"
if len(mdp) == 0 :
    print(mdp_trop_court.upper())
elif len(mdp) < 8:
    print(mdp_trop_court.capitalize())
elif mdp.isdigit():
    print("mdp doit contenir un string")
else:
    print("mdp valide")

```



LES FICHIER

Read Only ('r') : Open text file for reading. ...
Read and Write ('r+'): Open the file for reading and writing.
Write Only ('w') : Open the file for writing. ...
Write and Read ('w+') : Open the file for reading and writing.
Append Only ('a'): Open the file for writing.

```
chemin = "C:\Users\User\Desktop\Udemy\UdemyPython"  
print(chemin)--> #erreur de type (unicode error) car en  
# Unicode qui vise à donner à tout caractère de n'importe
```

pour régler le problème de l'antislash on utilise le raw string
chemin = r"C:\Users\User\Desktop\Udemy\UdemyPython"

r"\n" est la chaîne de deux caractères contenant '\' et
alors que "\n" est la chaîne contenant uniquement le caractère

```
ou  
chemin = r"C:/Users/User/Desktop/Udemy/UdemyPython"  
ou  
chemin = r"C:\\Users\\User\\Desktop\\Udemy\\UdemyPython"  
du caractère \n
```

lire le contenu d'un fichier :

more testfichier.txt #la commande more pour afficher le contenu

```
chemin = r"C:\Users\User\Desktop\Udemy\fichier\testfichier.txt"  
f = open(chemin, "r") # fonction open 2 args le chemin et le mode  
#donc on va lire le fichier et on va récupérer un objet fichier  
f.close() # toujours fermer un fichier déjà ouvert
```

```
#créer un bloc d'instruction avec cette façon de faire par exemple  
with open(chemin, "r") as f: # with open ce qu'on veut c'est  
    contenu = f.read()-->1- python #read() pour lire le contenu  
                                2- js
```

3- react

```
print(contenu)
```

```

    contenu = repr(f.read()) --> '1- python\n2- js\n3- js\n'
    # une forme de chaîne de caractères et ne pas interpréter

```

```

    contenu = f.readlines() --> ['1- python\n', '2- js\n', '3- js\n']
    # chaque ligne du fichier comme élément de la liste

```

```

    contenu = f.read().splitlines() --> ['1- python', '2- js', '3- js']
    # chaque ligne du fichier comme élément de la liste

```

Il se peut que sur Windows, vous ayez certains problèmes. Pour résoudre ces problèmes, vous devez ajouter l'encodage :

```

with open("fichier_txt", "r", encoding='utf-8') as f:
    contenu = f.read()
    pass # empêche le programme d'émettre un message d'erreur

```

écrire dans un fichier :

```

with open(chemin, "w") as f: # le mode write pour écrire
    f.write("bonjour") --> bonjour

```

```

with open(chemin, "a") as f: # mode append pour ajouter
    f.write("\nbonjour") --> bonjour

```

bonjour

```

lignes = ["- Pull the other one!\n",
          "- I am. And this my trusty servant Patsy.\n"]
# ouverture d'un fichier texte en ajout
with open("dialogues.txt", mode="a") as f:
    f.writelines(lignes) --> # créer un fichier dialogues.txt

```

Le standard Unicode définit les caractères et les points de code pour les lettres, syllabes, idéogrammes, signes de ponctuation, etc.

```
// les fichier JSON: un format de fichier et un module
```

JSON (Java Script Objet Notation) est un format de données structurées.(chaîne de caractère , liste , tableau, dict). Il est souvent utilisé pour sérialiser(la sérialisation d'une suite d'informations plus petites pour, par exemple, des données structurées et les échanger sur un réseau, etc).

Python possède deux types de données qui représentent un dictionnaire et les listes.

données json et leurs équivalents en Python.:

JSON - Python

true	True
false	False
string	str
number	int, float
array	list
object	dict
null	None

```
import json
```

```
chemin = r"C:\Users\User\Desktop\Udemy\fichier\testfichier.json"
```

ecrire dans un json: (ENCODAGE DE DONNÉES EN JSON) //encodage

```
with open(chemin, "w") as f: # with open ce qu'on veut créer
    json.dump("bonjour", f) -->"bonjour" # dump(les données)
    json.dump(list(range(4)), f) -->[0, 1, 2, 3]
    json.dump(list(range(4)), f, indent=4)---> [
        0,
        1,
        2,
        3
    ]
```

```

with open("testfichier.json", "w") as f:
    json.dump([1, 2, 3, {'4': 5, '6': 7}], f, separators=
    json.dump({'4': 5, '6': 7}, f ,sort_keys=True, i

```

lire les donnees d'un json: (DECODAGE DE DONNEES JSON)

```

with open(chemin, "r") as f:
    lire = json.load(f)
    print(lire)--> [0,1,2,3]

```

```

import json

```

```

employee = {
    "nom": "Marie Richardson",
    "id": 1,
    "recrutement": True,
    "department": "Ventes"
}

```

```

with open('data.json', 'w') as mon_fichier:
    json.dump(employee, mon_fichier)

```

dans le fichier data.json. Le contenu final de ce fichier
-->{"nom": "Marie Richardson", "id": 1, "recrutement": t

ajouter des donner dans un fichier json:

```

with open("testfichier.json", "a") as f:
    json.dump([4], f, indent=4)
[
    1,
    2,
    3,

][
    4

```

```
]
```

la syntaxe est fausse on peut pas avoir 2 objets a lire

Il est important de noter qu'un fichier JSON ne peut contenir qu'un seul objet.

Ainsi, si vous souhaitez stocker plusieurs chaînes de caractères, il faut utiliser une liste.

Ce fichier JSON est donc valide :

```
["Pascal", "Patrick"]
```

Mais pas celui-ci :

```
"Pascal"
```

```
"Patrick"
```

Dès que vous souhaitez stocker plusieurs objets, il faut utiliser une liste.

donc pour modifier (ajouter supprimer ..) des données dans un fichier JSON, on va le lire (mode read) puis les modifier dans notre programme.

dans le fichier json on a

```
[ 1,2,3]
```

```
with open("testfichier.json", "r") as f:
    donnees = json.load(f)
```

```
donnees.append(4)
```

```
with open("testfichier.json", "w") as f:
    json.dump(donnees, f, indent=4)
[1,2,3,4]
```

on a 3 modes:

r : read (pour lire le fichier)

w : write (pour écrire dans le fichier en écrasant ce qui est déjà dedans)

a : append (ajouter des données à celle qui sont déjà dans le fichier)

il faut toujours fermer le fichier qu'on a ouvert pour éviter de perdre le fichier.close(), ou avec la syntaxe (with openqui le

```
import json
chemin = r"C:\Users\User\Desktop\Udemy\fichier\testfichier\testfichier.json"
1
2
3
#quand python va lire le fichier il va placer le curseur au début
f = open(chemin, "r")
print(f.read())
f.seek(0) #si on remet pas le curseur au début avec seek(0)
#la fin du fichier (après le 3) il n'y a rien donc il va lire le contenu
contenu = f.read()
print(contenu)
f.close()
```

la méthode seek :

Déplacement au Début du Fichier :

```
python
Copy code
with open('mon_fichier.txt', 'r') as fichier:
    fichier.seek(0)
```

Déplacement à une Position Spécifique :

```
python
Copy code
with open('mon_fichier.txt', 'r') as fichier:
    fichier.seek(10) # Déplace le curseur à la position 10
```

Déplacement à la Fin du Fichier :

```
python
Copy code
```

```
with open('mon_fichier.txt', 'r') as fichier:
    fichier.seek(0, 2) # Le deuxième argument 2 indique
```

pour eviter ce probleme de lutilisation de read plusieurs fois
on va recupere le fichier dans une variable au tout debut

```
// read(10) lire jusqu'au 10eme caractere
```

Mode Description

'r' | C'est le mode par défaut. Il ouvre le fichier en lecture.

«w» | Ce mode ouvre le fichier en écriture. Si le fichier existe déjà, il est effacé.

'X' | Crée un nouveau fichier. Si le fichier existe déjà, l'opération échoue.

'a' | Ouvrez le fichier en mode ajout. Si le fichier n'existe pas, il est créé.

't' | C'est le mode par défaut. Il s'ouvre en mode texte.

«b» | Cela s'ouvre en mode binaire.

'+' | Cela ouvrira un fichier en lecture et en écriture.

use the open() function with the 'a' mode such as file = open('file.txt', 'a')

This function opens a file for writing, creating the file if it does not exist.



Gerer les chemins de fichier

Supprimer tous les fichiers d'un répertoire en Python

```
import os
dir = 'path/to/dir'
for f in os.listdir(dir):
    os.remove(os.path.join(dir, f))
```

```
import os, glob
dir = 'path/to/dir'
filelist = glob.glob(os.path.join(dir, "*"))
for f in filelist:
    os.remove(f)
```

```
dir = 'path/to/dir'
for file in os.scandir(dir):
```



```
os.remove(file.path)
```

supprimer les fichier dont l'extension est .py en python

```
import os
```

```
# Spécifiez le chemin du répertoire que vous souhaitez r  
repertoire = "/chemin/vers/le/repertoire"
```

```
# Parcourez le répertoire
```

```
for fichier in os.listdir(repertoire):
```

```
    if fichier.endswith(".py"):
```

```
        # Vérifiez que le fichier a bien l'extension ".p
```

```
        chemin_fichier = os.path.join(repertoire, fichier)
```

```
        # Supprimez le fichier
```

```
        os.remove(chemin_fichier)
```

```
        print(f"Le fichier {fichier} a été supprimé.")
```

```
print("Tous les fichiers .py ont été supprimés.")
```

renommer un fichier :

```
import os
```

```
os.rename('guru99.txt', 'career.guru99.txt')
```

```
with open("dialogues.txt", encoding="utf-8") as f:
```

```
    pass
```

Si le paramètre encoding n'est pas spécifié alors Python qui exécute le code (ce qui peut nuire à la portabilité utilisé par défaut par l'interpréteur, il faut utiliser

```
import locale
```

```
locale.getpreferredencoding()
```

```
'UTF-8'
```

on a 3 modules principaux pour gérer créer des fichiers :

module shutil : Le module shutil en Python est une bibliothèque de haut niveau sur les fichiers et les répertoires. Il permet de copier, déplacer, supprimer des fichiers et des répertoires, ainsi que pour archiver et extraire.

ex:

```
shutil.copy(source, destination) #copier un fichier
shutil.copytree(source, destination) # copier un dossier
shutil.move(source, destination)#déplacer un fichier
shutil.remove(file_path) #supprimer un fichier
dossier = Path("/Users/thibh/Documents/SiteWeb")
shutil.rmtree(dossier) #supprimer des dossier qui ne sont pas des dossiers
```

module glob : permet de scanner notre disque dur pour trouver des fichiers. Le module glob en Python est utilisé pour trouver tous les fichiers et répertoires utilisés par la ligne de commande Unix. Il offre un moyen simple de trouver des fichiers et des répertoires en utilisant des caractères génériques. La fonction glob prend un modèle de fichier et renvoie tous les chemins de fichiers qui correspondent à ce modèle.

renvoie tous les chemins de fichiers qui correspondent à ce modèle.

Astérisque (*) : correspond à zéro ou plusieurs caractères.
Point d'interrogation (?) : correspond exactement à un caractère.

ex :

```
path = '/chemin/vers/répertoire'
files = glob.glob(path + '/*.csv') # recupere les fichiers csv
```

```
files = glob.glob(path + '*.txt')
print(files)
```

```
liste_fichiers = glob.glob("*.py")
```

Il est possible d'effectuer une recherche récursive (c'est-à-dire une recherche à True et en utilisant la séquence ** pour inclure tous les sous-dossiers).

```
liste_fichiers = glob.glob("**/*.py", recursive=True)
```

Le modèle peut contenir des caractères génériques tels que `*` ou `*`. Par exemple, le modèle `*.txt` correspond à tous les fichiers de type `txt`.

module `fileinput`:

Ce module offre une classe auxiliaire et des fonctions pour lire ou écrire des fichiers d'une liste. Si vous n'avez besoin de lire ou d'écrire que dans un seul fichier, voir le module `file`.

```
with fileinput.input(files=('dialogues.txt', 'txt.txt')):  
    for line in f:  
        print(line) # récupérer les lignes des deux fichiers
```

module `tempfile` : Ce module crée des fichiers et répertoires temporaires.

module `os.path`: Le module `os.path` fournit des fonctions pour manipuler des chemins.

```
import os.path
```

```
chemin = os.path.abspath("monfichier.txt")  
print(chemin)  
# Si le répertoire de travail est /home/david/Documents  
# affiche /home/david/Documents/monfichier.txt
```

```
chemin = os.path.join("fichiers", "monfichier.txt")  
print(chemin)  
# Sous Windows affiche fichiers\monfichier.txt  
# Sous *nix ou MacOS, affiche fichiers/monfichier.txt
```

module `OS` : Ce module fournit une façon portable d'utiliser les fonctions du système d'exploitation. Si vous voulez simplement lire ou écrire un fichier, voir le module `file` ou `fileinput`. Pour la création de fichiers et de répertoires, voir le module `tempfile`.

```
import os
```

```

folder_path = "mon_dossier"
os.mkdir(folder_path)

folder_path = "/chemin/vers/le/repertoire"
files = os.listdir(folder_path)
for file in files:
    print(file)

file_path = "mon_fichier.txt"
absolute_path = os.path.abspath(file_path)
print(absolute_path)import os

# Accède à la variable d'environnement "HOME" sous Unix/
home_directory = os.environ["HOME"]
print(home_directory)import os

path_to_check = "/chemin/vers/un/fichier_ou_dossier"
if os.path.exists(path_to_check):
    print(f"{path_to_check} existe.")

# Joindre des chemins de manière portable
path1 = "/dossier1"
path2 = "fichier.txt"
full_path = os.path.join(path1, path2)

import shutil
folder_to_delete = "/chemin/vers/le/dossier_a_supprimer"
shutil.rmtree(folder_to_delete)

# Obtenir le répertoire parent d'un fichier
parent_directory = os.path.dirname(full_path)

# Obtenir le nom de fichier à partir du chemin

```

```

file_name = os.path.basename(full_path)

import os
chemin = r"/ESSAI.PY "
os.path.dirname(chemin) --> le dossier parent (c:/Users/

p = os.path.splitext(chemin) # nous retourne un tuple avec
print(p)-->('/ESSAI', '.PY ')

p = os.path.splitext(chemin)[1]
print(P)-->('.py')

```

/ce n'est pas pratique car on utilise des méthodes de chaînes
/de risque d'erreur car on manipule des strings pas des objets

on va utiliser la librairie pathlib qui va nous permettre

Connaître le répertoire de travail

Le répertoire de travail correspond au répertoire courant

Avec le module os

```

import os
os.getcwd()

```

Avec le module pathlib

```

import pathlib
pathlib.Path.cwd()

```

Connaître le répertoire utilisateur

Avec le module os

```

import os
os.environ['HOME']

```

Avec le module pathlib

```

import pathlib

```

```
pathlib.Path.home()
```

Copier un fichier

Avec le module shutil

```
import shutil
shutil.copy("monfichier.txt", "macopie.txt")
```

Supprimer un fichier

Avec le module os

```
import os
os.remove("monfichier.txt")
```

Avec le module pathlib

```
import pathlib
p = pathlib.Path("monfichier.txt")
p.unlink()
```

Supprimer un répertoire

Pour supprimer un répertoire, ce dernier doit être vide.

Avec le module os

```
import os
os.rmdir("monrepertoire")
```

Avec le module pathlib

```
import pathlib
p = pathlib.Path("monrepertoire")
p.rmdir()
```

Vérifier qu'un fichier existe

Avec le module os.path

```
import os.path
os.path.exists("monfichier.txt")
```

Avec le module pathlib

```
import pathlib
```

```
p = pathlib.Path("monfichier.txt")
p.exists()
```

```
crer ,supprimer un dossier avec os
folder_path = "/path/to/your/directory"
os.mkdir(folder_path)
os.rmdir(folder_path)
```

```
supprimer
shutil.rmtree(folder_path)
```

Historiquement, on utilisait des modules comme le module `shutil` pour la suppression et gestion des fichiers.

Le module `pathlib`, disponible depuis la version 3.4 de Python, permet de travailler sur un système d'exploitation, avec une syntaxe orientée objet.

librairie `Pathlib`: (Chemins de système de fichiers orientés objet)

```
>>> from pathlib import Path
>>> print(dir(Path)) # voir toutes les fonction qu'a la classe Path
```

Le module `pathlib` est un module de haut-niveau qui permet de travailler avec des chemins ou le répertoire désigné par ce chemin.

C'est un module tout-en-un qui facilite grandement le travail avec les chemins. L'élément central du module est la classe `Path`. Pour créer un objet `Path`, on va pouvoir faire toutes les opérations qu'on fait avec les chemins.

on va pouvoir faire toutes les opérations qu'on fait avec les chemins.

```
from pathlib import Path #path est une classe un objet
```

```
chemin = Path("/Users/User/Desktop/Udemy/exos/ESSAI.PY")
chemin.parent
WindowsPath('/Users/User/Desktop/Udemy/exos')
chemin.suffix --> .py
```

```
Path.home() --> WindowsPath('C:/Users/User') #recupérer le chemin du répertoire courant
```

```

Path.cwd()# //recupererle dossier courant

path = Path("/", "home", "david", "Documents", "monfichier.txt")
str(path)
'/home/david/Documents/monfichier.txt'
path.parts
('/', 'home', 'david', 'Documents', 'monfichier.txt')
path.root
'/'
path.drive
''
path.name
'monfichier.txt'
parent = path.parent
parent.parts
('/', 'home', 'david', 'Documents')
path.is_file()
True
path.is_dir()
False
path.parent.is_file()
False
path.parent.is_dir()
True

```

Obtenir le chemin absolu d'un fichier :

```

file_path = Path("mon_fichier.txt")
absolute_path = file_path.resolve()
print(absolute_path) # C:\Users\User\Desktop\Udemy\exos\

```

La classe Path possède la méthode open() qui accepte les chemins relatifs et absolus. Elle ouvre le fichier et retourne un objet de type File par l'objet lui-même :

```

from pathlib import Path

chemin = Path("dialogues.txt")

```



```

with chemin.open() as f:
    for ligne in f:
        print(ligne)

chemin = Path("dialogues.txt")
contenu = chemin.read_text()#read_text() qui ouvre le fi

Pour construire un chemin à partir d'un autre chemin: (c

    il suffit d'utiliser l'opérateur / qui est utilisé, nor
de la division, mais comme le séparateur universel de ch

chemin = Path("mondossier")
chemin_fichier = chemin / "mon fichier.txt"
chemin_fichier.parts
('mondossier', 'mon fichier.txt')

chemin_fichier = Path.home()
strch = str(chemin_fichier)
print(strch) --->C:\Users\User

chemin_fichier = Path.home() / "Documents" / "monfichier"
strch = str(chemin_fichier)
print(chemin_fichier)-->C:\Users\User\Documents\monfichi

On peut également utiliser la méthode joinpath sur un ok
path.joinpath("jugo.py")
Ça peut être pratique si vous avez par exemple une list
(grâce à l'unpacking et à l'opérateur splat *) :
home = Path.home() # PosixPath('/Users/thibh/')
dossiers = ['Projets', 'Django', 'blog']
home.joinpath(*dossiers)

home / "Projet" / "main.py".suffix # Ne fonctionne pas c

# Avec des parenthèses, ça fonctionne !

```

```
(home / "Projet" / "main.py").suffix
```

recuperer des infos sur un chemin:

```
from pathlib import Path
```

```
p = Path("/Users/thibh/Documents/index.html")
```

```
p.name      # "index.html"
```

```
p.parent    # "/Users/thibh/Documents"
```

```
p.stem      # "index"
```

```
p.suffix    # ".html"
```

```
p.suffixes  # 2 extentions
```

```
p.parts     # ("/", "Users", "thibh", "documents", "index.
```

Il existe également des méthodes qui permettent de vérifier

```
p.exists()   # True
```

```
p.is_dir()   # False
```

```
p.is_file()  # True
```

on peut donc mettre bout à bout plusieurs fois le même attribut
par exemple :

```
p = Path("/Users/thibh/Documents/index.html")
```

```
p.parent.parent # "/Users/thibh"
```

```
p.rename(dossier dans le quel on veut déplacer p)
```

Créer et supprimer des dossiers:

Pour créer un dossier, on peut utiliser la méthode `mkdir` si le dossier n'existe pas. Vous pouvez utiliser le paramètre `exist_ok` pour que pas qu'une erreur soit levée si le dossier existe déjà :

```
dossier = Path("/Users/thibh/Documents/SiteWeb")
```

```
dossier.mkdir() # Lève une erreur si le dossier existe
```

```
dossier.mkdir(exist_ok=True) # si le dossier existe déjà
```

Si vous souhaitez créer une hiérarchie de dossier qui n'existe pas

```
from pathlib import Path
```

```
# Le dossier SiteWeb et ses sous-dossiers n'existent pas  
dossier = Path("/Users/thibh/Documents/SiteWeb/sources/c")  
dossier.mkdir(parents=True) #pour crer tous les dossier
```

```
pour crer un fichier  
file = dossier / "readme.txt"  
file.touch
```

```
pour supprimer le fichier
```

```
file.unlink()
```

Pour supprimer un dossier, on utilise la méthode `rmdir` :

```
dossier = Path("/Users/thibh/Documents/SiteWeb")  
dossier.rmdir() #Cette méthode ne fonctionne que si le c
```

Si le dossier contient des fichiers ou d'autres sous-dossiers, c'est le seul cas de figure où nous sommes obligés de recourir à `shutil`.

```
import shutil  
from pathlib import Path
```

```
dossier = Path("/Users/thibh/Documents/SiteWeb")  
shutil.rmtree(dossier) # supprimer un hierarchie de dossier
```

Créer, lire et écrire dans un fichier:

```
fichier = Path("/Users/thibh/Documents/SiteWeb/index.html")  
fichier.touch() # On crée le fichier  
fichier.unlink() # On supprime le fichier
```

Pour écrire du contenu dans un fichier, on utilise la méthode `write_text`.

```
fichier = Path("/Users/thibh/Documents/SiteWeb/index.htm")
fichier.write_text("Accueil du site")
```

Il n'est pas obligatoire de créer le fichier au préalable

```
fichier.read_text()
```

c'est plus facile que de faire :

```
with open(fichier, "w") as f:
    f.write("Accueil du site")
```

scanner un dossier :

Pour récupérer tous les fichiers et dossiers à l'intérieur d'un dossier, on peut utiliser la méthode `iterdir` :

```
from pathlib import Path
for f in Path.home().iterdir():
    print(f.name)
```

```
from pathlib import Path
```

```
directory_path = Path("/chemin/vers/le/repertoire")
element_names = directory_path.listdir()
for name in element_names:
    print(name)
```

```
directory_path = Path("/chemin/vers/le/repertoire")
for element_path in directory_path.iterdir():
    print(element_path)
```

En résumé, `listdir()` renvoie une liste de noms de fichiers et de chaînes de caractères, tandis que `iterdir()` renvoie un itérateur sur chaque élément du répertoire, ce qui offre plus de souplesse.

On peut combiner cette méthode avec la méthode `is_dir` pour filtrer les dossiers :

```
dossiers = [d for d in Path.home().iterdir() if d.is_dir()]
```

Pour scanner un dossier avec un peu plus de flexibilité, le module `!`). On peut par exemple ne récupérer que les f

```
for f in Path.home().glob("*.png"):
    print(f.name)
```

Si vous souhaitez scanner un dossier de façon récursive(

```
for f in Path.home().rglob("*.png"):
    print(f.name) #afficher le nom des fichier avec.png
```

directement si vous souhaitez récupérer uniquement les c

Il suffit pour cela d'ajouter un slash à la fin du chemi

La classe `Path` possède également la méthode `glob()`:

```
liste_fichiers = list(pathlib.Path.cwd().glob("**/*.py"))
```

quelques cas concrets:

Ajouter un suffixe à un nom de fichier

```
from pathlib import Path
```

```
p = Path.home() / "image.png" # "/Users/thibh/image.png"
p.parent / (p.stem + "-lowres" + p.suffix) # "/Users/th
```

Il est possible de réaliser des opérations élémentaires soit avec le module `pathlib`. Les modules `os` et `shutil` s

Ils proposent surtout des fonctions alors que le module

trier des fichier selon leurs extension :

```

from pathlib import Path

dirs = {".jpg": "Images",
        ".png": "Images",
        ".mp4": "Videos",
        ".pdf": "Documents"}

tri_dir = Path.home() / "tri"
files = [f for f in tri_dir.iterdirs() if f.is_file()] #
#files est une liste avec tous les fichier du dossier tri
for f in files:
    # Si aucune correspondance n'est trouvé pour l'extension
    output_dir = tri_dir / dirs.get(f.suffix, "Autres")
    # avec la methode get si l'extension du fichier est trouvée
    # valeur associée à la clé sinon (ex si l'extension est .jpg)
    # a cette clé dans le dictionnaire qui est image
    get(key[, default])
    Renvoie la valeur de key si key est dans le dictionnaire
    output_dir.mkdir(exist_ok=True) #on va créer le dossier
    f.rename(output_dir / f.name)# rename pour déplacer

#on a un ensemble de fichiers dans notre dossier tri on va les trier

Créer les constantes d'un dossier avec file:

```

```

from pathlib import Path

SOURCE_FILE = Path(__file__).resolve() # resolve permet de
#la var __file__ retourne le chemin du script qu'on est en train d'exécuter
SOURCE_DIR = SOURCE_FILE.parent
ROOT_DIR = SOURCE_DIR.parent
DATA_DIR = SOURCE_DIR / "DATA"

avec le module os
import os
SOURCE_FILE = os.path.realpath(__file__)
SOURCE_DIR = os.path.dirname(SOURCE_FILE)#os.path.dirname renvoie le
ROOT_DIR = os.path.dirname(SOURCE_DIR)

```

```
DATA_DIR = os.path.join(SOURCE_DIR, "DATA")
```

Historiquement, on utilisait des modules comme le module `os` pour effectuer des opérations de création, suppression et gestion des fichiers.

Le module `pathlib`, disponible depuis la version 3.4 de Python, permet d'effectuer des opérations courantes sur un système d'exploitation, avec une syntaxe plus intuitive.

```
from pathlib import Path
import glob
```

```
BASE_DIR = Path(r"\Users\User\Downloads").glob("**/*")
print(BASE_DIR) #<generator object Path.glob at 0x0000001234567890>
```

```
print(list(BASE_DIR)) #une liste des fichiers que contient le dossier
```

retouver un fichier dans un dossier :

```
from pathlib import Path
```

```
# Spécifiez le répertoire dans lequel vous souhaitez rechercher le fichier
dossier = Path("C:\\Users\\User\\AppData\\Roaming\\Python\\Python34\\Scripts")
```

```
# Spécifiez le nom du fichier que vous recherchez
nom_du_fichier = "mon_module.py"
```

```
# Utilisez la méthode `glob` pour rechercher le fichier
fichiers_trouves = list(dossier.glob(f"*{nom_du_fichier}"))
```

```
# Vérifiez si des fichiers correspondants ont été trouvés
if fichiers_trouves:
```

```
    print("Fichiers trouvés :")
```

```
    for fichier in fichiers_trouves:
```

```
        print(fichier)
```

```
else:
```

```
    print("Aucun fichier correspondant n'a été trouvé.")
```



Les Dictionnaires : (on parle de tableaux associatifs ou de hashes dans d'autres langages)

Un ensemble de paires clé-valeur au sein duquel les clés sont uniques. Si cela arrive, cela ne va pas créer d'erreur mais votre dictionnaire sera modifié. Une paire d'accolades crée un dictionnaire vide : {}. Pour ajouter des accolades ajoute les valeurs correspondantes au dictionnaire.

pour chaque valeur stocker dans le dictionnaire on aura une fonction pour accéder à la valeur

stocker une valeur pour une clé et à extraire la valeur associée. Il est possible de supprimer une paire clé-valeur avec del. Si la clé est déjà utilisée, l'ancienne valeur associée à cette clé est supprimée. Si associée à une clé qui n'existe pas, une exception est levée.

Les dictionnaires sont des collections d'objets non-ordonnés.

Un dictionnaire est composé d'éléments et chaque élément est une paire clé-valeur.

Dans d'autres langages de programmation, on parle de tableaux associatifs ou de hashes.

Comme les listes, les dictionnaires sont des objets mutables et peuvent s'étendre selon vos besoins.

Un dictionnaire peut contenir des objets de n'importe quel type.

C'est grâce à ces caractéristiques que les dictionnaires sont utilisés dans des structures complexes où plusieurs éléments sont imbriqués les uns dans les autres.

créer un dictionnaire :

la fonction dict

```
# Dictionnaire dont les clés ne sont que des chaînes de caractères
d = {
    'spam': 'eggs',
```



```

        'knights': 'lumberjack',
        'bacon': 'sausage'
    }

print(d)

# Dictionnaire dont les clés sont des objets de différents types
# Une clé ne peut pas être un objet muable : on peut utiliser des objets immuables
d = {
    1: 'one',
    'deux': 2,
    (3, 4, 5): 'pas_de_soucis',
    9.9: 'nine_point_nine'
}

```

recuperer une valeur associer a une cle :

Alors qu'on accède aux éléments contenus dans une liste pour les dictionnaires, on utilise une clé.

On peut utiliser cette clé à l'intérieur de crochets []

1- avec les []

```

d = {
    9: {"prenom": "paul",
        "profession": "ingenieur"},
    8: {"prenom": "jugo",
        "profession": "architecte"}
}
print (d[9]["prenom"]) # paul (9 cest la cle du premier dictionnaire)
print(d[9]) # {'prenom': 'paul', 'profession': 'ingenieur'}

```

Avec les crochets, une erreur de type `KeyError` est levée si la clé n'existe pas. Alors que la méthode `get` vous retournera simplement `None` si la clé n'existe pas.

2-avec get

```

x = d.get(9)

```

```

print(o.get("prenom")) #paul

employees = {
    "id01": {"prenom": "Paul", "nom": "Dupont",
    "id02": {"prenom": "Julie", "nom": "Dupuit",
    "id03": {"prenom": "Patrick", "nom": "Ferrari"}
}

age_paul = employees.get("id01").get("age")
print(age_paul) #32

supprimer un dict :

del employees["id03"] # supprimer le dict 3

modifier la valeur d'une cle :

employees["id02"]["age"] = 26 # modifier l'age de julie
print(employees)

age_paul = employees.get("id01").get("age")# recuperer l'age de paul

si la cle n'existe pas :
d.get("nom", "la cle n'existe pas") # get(cle , valeur par defaut)

films = {"Le Seigneur des Anneaux": 12,
        "Harry Potter" : 9,
        "Blade Runner" : 7.5,
        }
prix = 0
for f in films:
    prix = prix + films.get(f)#12 9 7.5 En bouclant sur films
print(str(prix) + "€") # 28$

ou

for key in films: #Quand on boucle sur un dictionnaire avec for
    prix += films[key]

```

Ajouter et modifier des éléments :

```
d = {  
    'spam': 'eggs',  
    'knights': 'lumberjack',  
}
```

modifier une cle du dictionnaire

```
d['spam'] = 'ham' # Clé existe déjà, remplace la valeur
```

ajouter une cle au dictionnaire

```
d['bacon'] = 'sausage' # Nouvelle clé, on créer la paire
```

```
print(d)
```

```
d = {  
    'spam': 'ham',  
    'knights': 'lumberjack',  
    'bacon': 'sausage'  
}
```

supprimer un element :

```
del d["spam"]  
d.pop('knights')
```

```
d.clear() # {}  
del d
```

verifier si une cle existe dans le dico:

```
if "age" in d :  
    del ["age"]
```

Utiliser la méthode popitem pour supprimer un item aléatoire
une clé et sa valeur :

```
d = {  
    'spam': 'ham',
```

```

    'knights': 'lumberjack',
    'bacon': 'sausage'
}
item = d.popitem() # ('bacon', 'sausage') OU ('spam', 'lumberjack')

```

boucler sur un dictionnaire :
soit directement :

```

for cle in dicti :
    print(cle)

```

On pourrait utiliser la clé pour récupérer la valeur de

```

for key in d:
    print(key, d[key])

```

on a 2 methode utile:

```

d.keys() # retourne un liste qui contient les cle du dic
d.values() # dict_values(['ham', 'lumberjack', 'sausage'])
d.items()# on recupere la cle et sa valeur

```

```

for key in d.keys():
    print(key)

```

```

for value in d.values():
    print(value)

```

on combine les deux pour recuperer les aires cles valeur

```

for key, value in d.items():
    print(key, value)

```

```

#spam ham
#knights lumberjack
#bacon sausage

```

Parfois, vous aurez besoin qu'un dictionnaire soit initi

Pour faire ça, vous pouvez utiliser les defaultdict.

```
from collections import defaultdict
programming_languages = defaultdict(lambda: 'Python')
programming_languages['.js'] = 'JavaScript'
programming_languages['.php'] = 'PHP'

print(programming_languages['.js'])    # 'Javascript'
print(programming_languages['.php'])   # 'PHP'
```

Compréhension de dictionnaire : {clé: valeur for élément}

```
d = {
    'spam': 'ham',
    'knights': 'lumberjack',
    'bacon': 'sausage'
}
```

```
d_with_s = {k:v + 's' for k, v in d.items()} # Ajoute u
#{clé: valeur + 's' for clé, valeur in dictionnaire.item
print(d_with_s)
```

```
# Création d'un dictionnaire avec des carrés des nombres
carres = {x: x**2 for x in range(5)}
# Résultat : {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

```
# Création d'un dictionnaire avec des carrés des nombres
carres_pairs = {x: x**2 for x in range(10) if x % 2 == 0}
# Résultat : {0: 0, 2: 4, 4: 16, 6: 36, 8: 64}
```

verifier la presence d'une cle dans un dictionnaire :

```
d = {
    'spam': 'ham',
    'knights': 'lumberjack',
    'bacon': 'sausage'
}
```

```
print('bacon' in d) # True
print('ham' in d)   # False
```



gestion d'erreur avec les exception

Cela permet de gérer proprement les erreurs, de les traiter et ne s'arrête brutalement.

les exception et la programmation lbyl (look before you than persmission):

LYBL (Look Before You Leap) :

Cette approche consiste à vérifier si une condition est vraie avant d'effectuer une action. Vous effectuez une vérification préalable pour vous assurer que la condition est remplie. Si la vérification échoue, vous évitez de poursuivre avec une action qui pourrait générer une erreur. Cela peut conduire à un code plus verbeux avec de multiples vérifications.
ex :

```
liste = [1, 2, 3]

if len(liste) > 0:
    premier_element = liste[0]
    print(premier_element)
```

EAFP (Easier to Ask for Forgiveness than Permission) :

Cette approche consiste à effectuer une action et à gérer l'exception si elle se produit. Vous effectuez l'action d'abord et "demandez le pardon" si une erreur se produit. Cela peut rendre le code plus concis, mais il peut être moins sûr.
ex :

```
liste = [1, 2, 3]

try:
    premier_element = liste[0]
    print(premier_element)
except IndexError: #Un IndexError est généré lorsqu'une
```

```
print("La liste est vide ou n'a pas d'élément à l'ir
```

En général, le choix entre LYBL et EAFP dépend du langage et de la philosophie du code. Python, par exemple, privilégie souvent les exceptions.

les exceptions:

```
ex :
```

```
a = 0
```

```
b = 1
```

```
print(b/a) # ZeroDivisionError: division by zero
```

```
try:
```

```
    print(b/a)
```

```
except: # on gère tous les types d'erreurs
```

```
    print("division par zero impossible")
```

on peut spécifier le type d'erreur qu'on veut traiter:

```
a = 0
```

```
b = 3
```

```
try:
```

```
    print(b/a)
```

```
except ZeroDivisionError:
```

```
    print("division par zero impossible")
```

```
except TypeError:
```

```
    print("la var b n'est pas du bon type")
```

```
a = 0
```

```
b = "w"
```

```
try:
```

```
    resultt= b/a ## Bloc de code susceptible de lever une exception
```

```
except ZeroDivisionError: # Gestion de l'exception spécifique
```

```
    print("division par zero impossible")
```

```
except TypeError: # Gestion de l'exception spécifique(erreur de type)
```

```
    print("la var b n'est pas du bon type")
```

```
except Exception as e: # Gestion d'autres exceptions
```

```
    print("la var b n'est pas définie :{e}")
```

ou

```
else: #Le bloc else spécifie un code qui sera exécuté si  
    print(resultt)
```

```
finally: # execture le code dedans quoi qui arrive qu'il  
    print("fin du bloc")
```

- Le bloc try contient le code qui pourrait générer une e
- Le bloc except spécifie comment gérer une exception spé
- Vous pouvez avoir plusieurs blocs except pour gérer dif
- Le bloc except Exception as e capture toutes les autres
- Le bloc finally contient du code qui sera exécuté qu'il

```
entre = input("entrer le chemin d'un fichier a ouvrir :")
```

```
try:  
    with open(entre, "r") as f:  
        contenu_fichier = f.read()  
        print(contenu_fichier)  
except FileNotFoundError: #filenotfound cest lerreur qq  
    print("le fichier est introuvable")  
except UnicodeDecodeError as e:  
    print(f"impossible d'ouvrir le fichier: {e}")
```

La structure try...except permet de gérer les exceptions. Si une exception spécifique se produit, elle est capturée et un message d'erreur approprié est affiché. Si vous souhaitez vous pouvez ajouter plusieurs clauses except. on teste le code on voit quel erreur nous est retourner



Les Fonctions

Les fonctions en Python sont des blocs de code réutilisés. Elles permettent de diviser un programme en morceaux plus petits, ce qui facilite la lecture, la maintenance et la réutilisation du code.

Une fonction permet de regrouper une suite d'instructions sous un nom unique.

Fonctions Sans Valeur de Retour :

on les utilise que pour exécuter le code qu'elles contiennent. Les fonctions sans valeur de retour, également appelées fonctions sans renvoyer explicitement une valeur. Elles utilisent l'écran ou la modification d'objets globaux.

ex : `print()`

Les fonctions avec valeur de retour renvoient une valeur calculée à partir de leurs paramètres. Elles utilisent l'instruction `return` pour spécifier la valeur à renvoyer.

Créer une fonction `format_paragraph`:

on utilise le mot clé `def` : (defined)

```
def nom_dela_fonction():  
    print("jugo")
```

une fois qu'une fonction est définie, on peut l'appeler. Il suffit de suivre de parenthèses pour exécuter le code qu'elle contient.

```
nom_dela_fonction() #jugo
```

Retourner une valeur:

retourner une valeur dans une fonction grâce au mot clé `return`. vous pouvez avoir plusieurs instructions `return`, mais une seule de laquelle est atteinte en premier. L'exécution de la fonction s'arrête à la première instruction `return`.

est exécutée.

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

```
resultat = somme(10, 5)  
print(resultat)
```

```
def somme(a, b):  
    return a + b  
    print(a + b) # donc ce print est inatteignable  
somme(10, 5)
```

ex :

```
def verif_fichier():  
    if os.path.exists("nomfichier")  
        return true  
    else:  
        return false
```

on la reecris plus simplement

```
def verif_fichier():  
    if os.path.exists("nomfichier")  
        return true #si le fichier existe en retourne t  
    return false # si le fichier existe pas donc on va p
```

meme sans utiliser return une fonction retourne toujours

```
def test()  
    pass
```

```
a = test() #None
```

```
def ma_fonction():  
    return 5  
    print("Fin de la fonction")
```

```
ma_fonction() # ca va rien afficher (le return termine l  
#, on n'affiche donc rien.)
```

Paramètres de Fonction : Les paramètres sont des valeurs pour qu'elle les utilise pendant son exécution comme des avec des valeurs par défaut.

ex :

```
def bienvenue(nom, message="Bonjour"): # la fonction bienvenue
    print(f"{message}, {nom}!")
```

```
bienvenue("Alice") # Affiche "Bonjour, Alice!"
#quand on appelle la fonction on lui passe un argument
bienvenue("Bob", "Salut") # Affiche "Salut, Bob!"
```

les paramètres : sont les variables qui sont définies dans la fonction
les arguments : sont les valeurs qu'on va envoyer à ses paramètres
l'argument va être récupéré par la fonction et l'utiliser

si on passe pas d'argument à une fonction qui a des paramètres
il existe des paramètres par défaut:

ex:

```
def affiche(msg= "jugo"):
    print(f"bnjr {msg}")
```

```
affiche() # bnjr jugo si on lui passe pas d'argument elle affiche "jugo"
affiche("adjoud") # bnjr adjoud si on lui passe un argument elle affiche "adjoud"
```

```
def ma_fonction(parametre1, parametre2):
    # Bloc de code de la fonction
    resultat = parametre1 + parametre2
    return resultat
```

```
resultat = ma_fonction(10, 20) # respectivement 10 à parametre1 et 20 à parametre2
print(resultat) # Affiche 30
```

ou

```
resultat = ma_fonction(paremetre2 = 10, parametre1 = 20)
```

```
def add(a=1, b=2, c=3):  
    return a + b
```

```
add(c=5, b=6) #7
```

definir les fonction au bon endroit:

```
test() # erreur car test pas encore definie
```

```
def test():  
    affichebnjr()# pas d'erreur car test on l'appelle apres avoir defini
```

```
def affichebnjr():  
    print("bnjr")
```

```
test() #bnjr
```

variable local et globale :

notion d'espace

Les variables déclarées à l'intérieur d'une fonction ont une portée locale, ce qui signifie qu'elles ne sont accessibles que dans cette fonction. Les variables déclarées en dehors de toutes les fonctions ont une portée globale, ce qui signifie qu'elles peuvent être utilisées partout.

```
variable_globale = 10
```

```
def ma_fonction():  
    variable_locale = 5  
    print(variable_globale) # Accessible (portée globale)  
    print(variable_locale) # Accessible (portée locale)
```

```
ma_fonction()  
print(variable_globale) # Accessible  
print(variable_locale) # Erreur, variable_locale n'est pas définie
```

```
def fonct():
    a = 2 # var locale accessible que dans cette fonction
a = "q" # var globale
meme avc le meme nom ce sont 2 var differentes
```

```
a = 5
def add(a=1, b=2):
    return a + b
```

```
add(a) # 5
```

on a tjr un espace globale mais on peut tjr crer plusieurs espaces locaux de l'espace globale c l'espace globale

les fonctions globals et locals :

s

La Fonction locals() :

La fonction locals() renvoie un dictionnaire contenant toutes les variables de la portée actuelle, c'est-à-dire celles qui sont définies localement. locals() renvoie un dictionnaire qui est une copie des variables locales.
ex:

```
def ma_fonction():
    variable_locale = "Python"
    print(locals())
```

```
ma_fonction() #{'variable_locale': 'Python'}
```

La Fonction globals() :

La fonction globals() renvoie un dictionnaire contenant toutes les variables du programme, y compris celles qui sont définies en dehors du module courant. Dans l'espace globale, il n'y a pas de chose qui soit locale.
ex:

```
variable_globale = 42
```

```
def functglob():
```

```

    variable_locale = "Bonjour"
    print(globals())

fonction() #{'__name__': '__main__', '__doc__': None, '_

variable_globale = 42
def fonction():
    variable_locale = "Bonjour"
    print(locals()) # {'variable_locale': 'Bonjour'}
    print(a) # 5 la variable a est dans l'espace globale

a = 5
fonction()

{python va d'abord voir dans l'espace locale si il ya qlq

linstruction global :

def getcomm(note):
    if note > 15 :
        commentaire = "bravo"
    elif 10 < note < 15:
        commentaire = "peut mieu faire"
    elif 5 < note < 10:
        commentaire = "attention"

commentaire = "tout faux"
getcomm(14)
print(commentaire) # tout faux #la variable globale comm
# la variable commentaire dans la fonction est une var l
#variable qui est en dehors de la fonction

une solution qui est deconseiller est d'utiliser linstruc
ce qui nous rend la fonction impure car elle modifie des
def getcomm(note):
    global commentaire # on dit a la fonction qu'on veut
    if note > 15 :

```

```

        commentaire = "bravo"
    elif 10 < note <15:
        commentaire = "peut mieu faire"
    elif 5 <note <10:
        commentaire = "attention"

commentaire = "note"
getcomm(14)
print(commentaire) # peut mieu faire

```

on fais plutot ca :

```

def getcomm(note):
    if note > 15 :
        comm = "bravo"
    elif 10 < note <15:
        comm = "peut mieu faire"
    elif 5 <note <10:
        comm = "attention"
    else:
        comm = "tu a tout faux"

    return comm

```

```

commentaire = getcomm(14)
print(commentaire) # peut mieu faire

```

le passage par reference :

Objets Immuables (Passage par Valeur) :

Les objets immuables, tels que les nombres, les chaînes valeur dans les fonctions. Cela signifie que lorsque vous copiez la valeur, une copie de la valeur est transmise, et toute modification ne modifie pas l'objet d'origine.

ex:

```

def modifier_valeur(x):
    x = 10 # Cette modification n'affecte pas la variable

```

```

a = 5
modifier_valeur(a)
print(a)  # Affiche 5, car 'a' n'a pas été modifié dans

```

Objets Mutables (Passage par Référence) :

Les objets mutables, tels que les listes et les dictionnaires. Cela signifie que lorsque vous passez un objet mutable à une fonction, vous passez l'objet d'origine, et toute modification à l'intérieur de la fonction affecte l'objet d'origine.

```

def modifier_liste(ma_liste):
    ma_liste.append(4)  # Cette modification affecte la liste d'origine

liste = [1, 2, 3]
modifier_liste(liste)
print(liste)  # Affiche [1, 2, 3, 4], car la liste a été modifiée

```

pour régler ce pb:

```

def modifier_liste(param):
    copy = param.copy()  # on copie la liste qu'on a envoyée
    copy.append(4)
    print(copy)

```

```

liste = [1, 2, 3]
modifier_liste(param =liste)  # [1, 2, 3, 4]
print(liste)  # [1, 2, 3]

```

Si vous souhaitez éviter de modifier un objet mutable par défaut, vous pouvez passer une copie de l'objet à l'intérieur de la fonction. En revanche, ne modifiez pas les objets directement dans la fonction, car toute modification est locale au lieu de modifier l'objet d'origine.

L'ordre des paramètres dans une fonction a son importance. Une valeur par défaut après un paramètre qui en a une, provoquera une erreur.


```

ex :
def multiplicateur_mot(mult=5, mot): #cest faux
    return mot * mult
exos sur les fonctions :

def saluer(nom):
    return f"bonjour {nom}"

salut = saluer("Patrick")
print(salut)
ou
def saluer(nom):
    return f"bonjour {nom}"

salut = saluer("Patrick")
print(salut)

def addition(a, b):
    c = a + b
    print(c)

resultat = addition(5, 10) #resulat = la valeur que retourne la fonction
print(resultat)#none
#. La fonction addition(a, b) ne renvoie pas explicitement de valeur (sans
#return), c'est pourquoi la variable resultat dans print() affiche None.

```

Une fonction en Python qui n'a pas d'instruction return, lorsque vous appelez print(), elle effectue une opération explicite.

Dans votre code initial, vous aviez une fonction addition. Lorsque vous avez appelé cette fonction et stocké le résultat dans une variable, vous avez obtenu None, car la fonction n'a pas renvoyé de valeur.

La fonction sert à calculer la somme de deux valeurs. Ici, elle retourne le résultat de cette addition.

Pour retourner une valeur dans une fonction, on utilise

```
def addition(a, b):
    c = a + b
    return c

resultat = addition(5, 10) # resultat = la valeur que retourne la fonction
print(resultat)
```

Les annotations de type : (type hinting)

une nouveauté de la version 3.5 de Python. pour éviter les erreurs de typage car python est un langage dynamique (les types des variables ne sont pas vérifiés à l'exécution).

Les annotations de type en Python sont une fonctionnalité qui permet d'annoter une variable, d'un argument de fonction ou de la valeur de retour d'une fonction.

Ces annotations ne sont pas obligatoires, mais elles aident à détecter les erreurs de typage et elles peuvent être utilisées par des outils d'analyse statique.

Annotations de type pour des variables :

```
x: int = 5
name: str = "John"
```

Annotations de type pour des paramètres de fonction et des valeurs de retour :

```
def add(a: int, b: int = 0) -> int:
    return a + b
```

```
def greet(name: str) -> str:
    return "Hello, " + name
```

Annotations de type pour des listes, des dictionnaires, des tuples, des ensembles :

```
from typing import List, Dict
```

```
numbers: List[int] = [1, 2, 3, 4]
person: Dict[str, str] = {"name": "Alice", "age": "30"}
```

```
def add(a: list[int], b: int = 0) -> list[int]:
```

```

    a.append(b)
    return a
r = add([1,2])
print(r)

```

utiliser le module mypy pour verifier le code

Mypy est un outil d'analyse statique de types pour Python de type

Les erreurs de type peuvent entraîner des bugs et des crashes. Mypy peut les repérer avant même que vous n'exécutiez votre code, plus tôt dans le processus de développement.

```

installer : python3.10 -m pip install mypy
executer  : mypy lecheminversle fichierqu'onveuxverfier
ex : le fichier ce trouve dans le bureau
cd desktop
mypy fichier.py

```

Les Modules :

Si vous avez l'intention de partager votre module avec d'autres développeurs, vous pouvez le publier sur le Package Index (PyPI) ou de le partager via d'autres moyens.

```

import module ex: import random on peut utiliser toutes les fonctions de random
ex : random.uniform(2.5)
importer une fonction specifique du module
from module import fonction ex: from random import uniform
#pas recommander car si on cree une variable dans l'espace de nom, la fonction ne sera plus disponible
uniform(2.5)

```

```

from random import * #deconseiller car on importe toutes les fonctions de random

```

créer notre propre module :

il suffit de créer un fichier.py et de l'importer

ex on crée le fichier : mon_module.py (ne pas utiliser de

a = 5

et un autre : essai.py

dans le fichier essai.py on importe le module qu'on a créé

```
import mon_module
```

```
print(mon_module.a) # 5
```

mon_module.py :

```
def addition(a : int , b : int) -> int:
```

```
    return a+b
```

```
def soustraction(a, b):
```

```
    return a - b
```

```
print(addition(2,3)) # 5
```

essai.py : import mon_module : si on exécute ce fichier

```
from mon_module import addition, soustraction
```

```
result_add = addition(5, 3)
```

```
result_sub = soustraction(10, 4)
```

la variable `__name__` : La variable `__name__` permet de déterminer

ou s'il est importé en tant que module dans un autre script

on la déclare pas elle prend sa valeur automatiquement en fonction

ou comme module `__init__ == "nomdumodule"`

Si un script Python est exécuté en tant que programme principal

la variable `__name__` sera définie comme `"__main__"`. Par conséquent

```
if __name__ == "__main__": # Ce code sera exécuté si le script
```

Cela permet d'encapsuler du code qui doit être exécuté uniquement

comme module

un script Python est importé en tant que module dans un script Python, le nom du module est le nom du script lui-même (c'est-à-dire le nom du module). Par exemple, si vous exécutez un script nommé `mon_module.py`, le nom du module sera défini comme `mon_module`.

ex :

```
# mon_module.py
```

```
def my_function():  
    print("Fonction de mon module")
```

```
if __name__ == "__main__":  
    print("Exécuté en tant que programme principal") # si le script est exécuté directement  
else:  
    print("Importé en tant que module") # si le script est importé dans un autre script
```

Lorsque vous exécutez `mon_module.py` directement, le code `if __name__ == "__main__":` est exécuté. Si vous importez `mon_module` dans un autre script, le code `else:` est exécuté.

```
#mon_module.py  
def addition(a : int , b : int) -> int:  
    return a+b  
# print(addition(2,3))  
if __name__ == "__main__":  
    print(addition(2 ,3))  
else:  
    print(__name__)  
#quand on execute ce script on a 5  
#essai.py  
import mon_module  
#quand on execute ce script n a le nom du module (mon_module)
```

python Path :

`PYTHONPATH` est une variable d'environnement utilisée pour spécifier les chemins des répertoires contenant des modules et des packages lorsqu'il est exécuté. Cette variable vérifie les chemins où Python recherche les modules lorsque vous utilisez `import`.

```

on peut afficher cette variable avec le module sys
import sys
from pprint import pprint

pprint(sys.pypath)
['c:\\Users\\User\\Desktop\\testpyudemy', #le dossier courant
 'C:\\Program Files\\Python310',
 ...
 'C:\\Users\\User\\AppData\\Roaming\\Python\\Python310\\
 'C:\\Users\\User\\AppData\\Roaming\\Python\\Python310\\
 'C:\\Program Files\\Python310\\lib\\site-packages']
#si on veut rendre un module disponible on peut le rajouter
#et des packages lorsqu'il est exécuté

ajouter un dossier a la variable path de python qui est
import sys

sys.path.append(r"C:\Users\User\Desktop\mes_modules")
print(sys.path)

cd desktop
ls #liste les dossier
cd mes_modules
pwd #le chemin courant
echo module_test.py # pour creer un fichier vide
code module_test.py # pour l'ouvrir dans vscode

on peut mettre le chemin de ce module dans nos variable

actualiser un module :
import importlib
importlib.reload(NOMDUFMODULE)

les package:

se sont des dossier qui comportent un ou plusieurs modules
un dossier mes_modules qui contient 2 fichiers (modules)

```

```
module1.py
module2.py
ex : from mes_modules import module1
```

le fichier `__init__`:

(dans les versions inferieur a 3.3 de python on est obl
Lorsque Python rencontre un répertoire contenant un fich
Cela signifie que le répertoire peut contenir des modul
que modules du package.

Le code contenu dans le fichier `__init__.py` est exécuté
effectuer des initialisations spécifiques au package ou
de l'importation du package.

. Vous pouvez ajouter davantage de modules, de fichiers
dans des répertoires et en utilisant le fichier `__init__`

Le module Logging :
j

req et res :