

Python : STDLIB

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`



- 1 Introduction
- 2 Fonctions mathématiques
 - math
 - random
 - operator
- 3 Quelques autres types
 - string
 - decimal
 - fraction
 - array
 - typing

4 Manipulation des dates

- `datetime`
- `pytz`
- `zoneinfo`
- `time`

5 Fonctions sur les itérables

- `functools`
- `functional`
- `collections`

6 Fonctions d'interaction avec le système d'exploitation (OS)

- `os`
- `glob`
- `shutil`
- `sys`

7 Autres modules

- `importlib`
- `copy`
- `zlib`
- `keyword`

8 Conventions

STDLIB

- Librairie standard de **Python**
- Collection de modules et de packages intégrés : disponibles dès l'installation de **Python**
- Conçu pour simplifier le développement de logiciels
- Offrant des outils pour des tâches courantes

Quelques modules **STDLIB** : fonctions mathématiques

- `math` pour les fonctions arithmétiques
- `random` pour la génération de nombres aléatoires
- `decimal` pour une meilleure manipulation de nombres décimaux
- `statistics` pour les fonctions statistiques de base
- ...

Quelques modules **STDLIB** : manipulation de données

- `datetime` : manipulation des dates et des heures.
- `json` : encodage et décodage de données **JSON**.
- `csv` : lecture et écriture de fichiers au format **CSV**.
- `zlib` : compression et décompression de données.
- ...

Quelques modules **STDLIB** : fonctionnalités de Fichiers et d'OS

- `glob` pour la recherche de fichiers/répertoires
- `os` pour réaliser des opérations sur le système d'exploitation
- `os.path` pour manipuler les chemins de fichiers
- `pathlib` pour la manipulation orientée objet des chemins de fichiers
- `sys` pour effectuer des opérations d'entrée et sortie plus personnalisées
- `shutil` pour réaliser des opérations sur les fichiers
- ...

Quelques modules **STDLIB** : réseau et communication

- `socket` : Interface de bas niveau pour les communications réseau
- `http` : Modules pour travailler avec **HTTP**
 - `http.client` pour le client
 - `http.server` pour le serveur
- `urllib` : Collection de modules pour travailler avec les URLs.
- ...

Quelques modules **STDLIB** : Qualité des Applications (**QA**)

- unittest
- doctest

© Achref EL M...

Python

Quelques modules **STDLIB** : Qualité des Applications (**QA**)

- `unittest`
- `doctest`

Quelques modules **STDLIB** : développement d'interface graphique (**GUI**)

tkinter : développement d'applications desktop.

Fonctions mathématiques prédéfinies (**Built-in functions**)

- `abs(x)` : retourne la valeur absolue de x
- `pow(x, y)` : retourne x puissance y
- `max(x, y)` : retourne le max de x et y
- `min(x, y)` : retourne le min de x et y
- `round(x)` : retourne l'arrondi de x

Python

Fonctions mathématiques prédéfinies (Built-in functions)

- `abs(x)` : retourne la valeur absolue de x
- `pow(x, y)` : retourne x puissance y
- `max(x, y)` : retourne le max de x et y
- `min(x, y)` : retourne le min de x et y
- `round(x)` : retourne l'arrondi de x

Exemple

```
print(max(1, abs(-3), 2))  
# affiche 3
```

Python

Fonctions mathématiques nécessitant l'importation du module `math`

- `sqrt(x)` : retourne la racine carré de `x`
- `trunc(x)` : retourne la partie entière de `x`
- `fsum(iterable)` : retourne la somme de tous les éléments de `iterable`
- `factorial(x)` : retourne la factorielle de `x`
- `floor(x)` et `ceil(x)` : retournent l'arrondi de `x`
- ...

© ADP

Python

Fonctions mathématiques nécessitant l'importation du module `math`

- `sqrt(x)` : retourne la racine carré de `x`
- `trunc(x)` : retourne la partie entière de `x`
- `fsum(iterable)` : retourne la somme de tous les éléments de `iterable`
- `factorial(x)` : retourne la factorielle de `x`
- `floor(x)` et `ceil(x)` : retournent l'arrondi de `x`
- ...

Exemple

```
import math

print(math.sqrt(25))
# affiche 5.0
```

Python

Exemple avec `math.floor(x)`, `math.ceil(x)` **et** `round(x)`

```
print(round(1.9))
# affiche 2
print(round(1.5))
# affiche 2
print(round(1.4))
# affiche 1
print(math.ceil(1.9))
# affiche 2
print(math.ceil(1.5))
# affiche 2
print(math.ceil(1.4))
# affiche 2
print(math.floor(1.9))
# affiche 1
print(math.floor(1.5))
# affiche 1
print(math.floor(1.4))
# affiche 1
```

Python

Pour extraire la partie entière et la partie décimale, on utilise la fonction `modf` (modulus and fraction)

```
frac, ent = math.modf(2.5)
```

```
print(frac)  
# affiche 0.5
```

```
print(ent)  
# affiche 2.0
```

Exercice

Écrire une fonction `equation_second_degre(a, b, c)` qui permet de résoudre une équation de second degré ($ax^2 + bx + c = 0$)

- $\Delta = b^2 - 4ac$
- Si $\Delta < 0$: pas de solution
- Si $\Delta = 0$: une solution $-b/2a$
- Si $\Delta > 0$: deux solutions $(-b - \sqrt{\Delta})/2a$ et $(-b + \sqrt{\Delta})/2a$

Résultat attendu

```
print(equation_second_degre(1, 1, -2))  
# affiche (-2.0, 1.0)  
  
print(equation_second_degre(1, 2, 1))  
# affiche -1.0  
  
print(equation_second_degre(2, 3, 5))  
# affiche pas de solution
```

Python

Solution

```
import math

delta = lambda a, b, c : pow(b, 2) - 4 * a * c

def equation_second_degre(a, b, c):
    d = delta(a, b, c)
    if d < 0:
        return "pas de solution"
    elif d == 0:
        return -b / (2 * a)
    else:
        return (-b - math.sqrt(d)) / (2 * a), (-b + math.sqrt(d)) / (2 * a)

print(equation_second_degre(1, 1, -2))
# affiche (-2.0, 1.0)

print(equation_second_degre(1, 2, 1))
# affiche -1.0

print(equation_second_degre(2, 3, 5))
# affiche pas de solution
```

Python

Fonctions mathématiques nécessitant l'importation du module `random`

- `random()` : retourne un nombre réel aléatoire entre 0 et 1
- `randint(x, y)` : retourne un nombre entier aléatoire entre `x` et `y` (`x` et `y` inclus)
- `choice(iterable)` : retourne un élément aléatoire appartenant à `iterable`
- ...

© Achret LL

Python

Fonctions mathématiques nécessitant l'importation du module `random`

- `random()` : retourne un nombre réel aléatoire entre 0 et 1
- `randint(x, y)` : retourne un nombre entier aléatoire entre `x` et `y` (`x` et `y` inclus)
- `choice(iterable)` : retourne un élément aléatoire appartenant à `iterable`
- ...

Exemple

```
import random

print(random.randint(5, 10))
# affiche un nombre entre 5 et 10
```

Python

Le module `operator` fournit un ensemble de fonctions

- qui correspondent aux opérations intrinsèques du langage, telles que les opérations arithmétiques, les comparaisons et les opérations sur les séquences.
- souvent utilisées comme callback par d'autres modules comme `functools`.

Python

Opérations arithmétiques fournies par `operator`

- `operator.add(a, b)` : **Additionne a et b** (équivalent à $a + b$).
- `operator.sub(a, b)` : **Soustrait b de a** (équivalent à $a - b$).
- `operator.mul(a, b)` : **Multiplie a et b** (équivalent à $a * b$).
- `operator.truediv(a, b)` : **Divise a par b** (équivalent à a / b).

© Achret

Python

Opérations arithmétiques fournies par `operator`

- `operator.add(a, b)` : Additionne `a` et `b` (équivalent à `a + b`).
- `operator.sub(a, b)` : Soustrait `b` de `a` (équivalent à `a - b`).
- `operator.mul(a, b)` : Multiplie `a` et `b` (équivalent à `a * b`).
- `operator.truediv(a, b)` : Divise `a` par `b` (équivalent à `a / b`).

Exemple

```
import operator

print(operator.add(2, 5))
# affiche 7
```

Python

Opérations de comparaison fournies par `operator`

- `operator.eq(a, b)` : Vérifie si `a` est égal à `b` (équivalent à `a == b`).
- `operator.ne(a, b)` : Vérifie si `a` est différent de `b` (équivalent à `a != b`).
- `operator.lt(a, b)` : Vérifie si `a` est inférieur à `b` (équivalent à `a < b`).
- `operator.le(a, b)` : Vérifie si `a` est inférieur ou égal à `b` (équivalent à `a <= b`).
- `operator.gt(a, b)` : Vérifie si `a` est supérieur à `b` (équivalent à `a > b`).
- `operator.ge(a, b)` : Vérifie si `a` est supérieur ou égal à `b` (équivalent à `a >= b`).

© ADP

Python

Opérations de comparaison fournies par `operator`

- `operator.eq(a, b)` : Vérifie si `a` est égal à `b` (équivalent à `a == b`).
- `operator.ne(a, b)` : Vérifie si `a` est différent de `b` (équivalent à `a != b`).
- `operator.lt(a, b)` : Vérifie si `a` est inférieur à `b` (équivalent à `a < b`).
- `operator.le(a, b)` : Vérifie si `a` est inférieur ou égal à `b` (équivalent à `a <= b`).
- `operator.gt(a, b)` : Vérifie si `a` est supérieur à `b` (équivalent à `a > b`).
- `operator.ge(a, b)` : Vérifie si `a` est supérieur ou égal à `b` (équivalent à `a >= b`).

Exemple

```
import operator

print(operator.gt(2, 5))
# affiche False
```

Python

string

Utilisé pour fournir des constantes et des fonctions utiles pour manipuler des chaînes de caractères

- **Constantes utiles** : comme `string.ascii_letters`, `string.digits`, `string.punctuation`...
- **Méthodes de formatage** : fournit des fonctions pour formater des chaînes de caractères, comme `string.format()`, `string.Template`...

Python

Quelques problèmes avec float

```
import string
```

© Achref EL MOUELHI ©

Python

Quelques problèmes avec float

```
import string
```

Exemple avec les constantes

```
print(string.ascii_lowercase)
# affiche abcdefghijklmnopqrstuvwxyz

print(string.digits)
# affiche 0123456789

print(string.ascii_letters)
# affiche abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Python

decimal

- Module **Python** permettant de manipuler des nombres à virgules dont le calcul est plus exacte en comparaison avec le type `float`
- Type conçu principalement pour les utilisateurs et pas pour la machine

© Achref EL MOU

Python

decimal

- Module **Python** permettant de manipuler des nombres à virgules dont le calcul est plus exacte en comparaison avec le type `float`
- Type conçu principalement pour les utilisateurs et pas pour la machine

Quelques problèmes avec `float`

```
print(0.1 + 0.1 + 0.1 - 0.3)
# affiche 5.551115123125783e-17
```

```
print(1.1 + 2.2)
# affiche 3.3000000000000003
```

Python

On peut construire un décimal à partir d'un entier ou une chaîne de caractères

```
from decimal import *

print(Decimal(3))
# affiche 3

print(Decimal("3.33"))
# affiche 3.33

print(Decimal(str(2.0 ** 2)))
# affiche 4.0
```

© Achret L

Python

On peut construire un décimal à partir d'un entier ou une chaîne de caractères

```
from decimal import *

print(Decimal(3))
# affiche 3

print(Decimal("3.33"))
# affiche 3.33

print(Decimal(str(2.0 ** 2)))
# affiche 4.0
```

On peut aussi construire un décimal à partir d'un float (mais il est conseillé de passer par les chaînes)

```
print (Decimal(3.3))
# affiche 3.29999999999999982236431605997495353221893310546875

print (Decimal(str(3.3)))
# affiche 3.3
```

Python

On peut modifier la précision

```
x = Decimal('2.4444')  
y = Decimal('3.4321')
```

```
print(x + y)  
# affiche 5.8765
```

```
getcontext().prec = 3  
print(x + y)  
# affiche 5.88
```

```
getcontext().prec = 2  
print(x + y)  
# affiche 5.9
```

```
getcontext().prec = 1  
print(x + y)  
# affiche 6
```

Python

fraction

- Module **Python** permettant de manipuler des nombres rationnels
- Construction possible depuis une paire d'entiers, un autre nombre rationnel, ou une chaîne de caractères.

Python

Exemples de construction de fraction

```
from decimal import Decimal
from fractions import Fraction

print(Fraction(5, 2))
# affiche 5/2

print(Fraction(2.5))
# affiche 5/2

print(Fraction(Decimal("2.5")))
# affiche 5/2

print(Fraction(5 / 2))
# affiche 5/2

print(Fraction("5/2"))
# affiche 5/2
```

Python

Pour extraire le numérateur ou le dénominateur

```
x = Fraction(2.5)

print(x.numerator)
# affiche 5

print(x.denominator)
# affiche 2
```

Les opérateurs arithmétiques peuvent être appliqués sur les fractions

```
f1 = Fraction(1, 2)
f2 = Fraction(3, 4)

# Addition
result_add = f1 + f2
print(result_add)
# affiche 5/4

# Soustraction
result_sub = f1 - f2
print(result_sub)
# affiche -1/4

# Multiplication
result_mul = f1 * f2
print(result_mul)
# affiche 3/8

# Division
result_div = f1 / f2
print(result_div)
# affiche 2/3
```

Python

Tableaux statiques : `array`

- Structure de données acceptant plusieurs valeurs de même type
- Type à spécifier à la création du tableau en utilisant le type code
- Seuls les types primitifs suivants sont acceptés : caractères, entiers et flottants

Python

Quelques types codes

- `b` : pour les caractères signés (byte : nombre compris entre -128 et 127)
- `B` : pour les caractères non-signés (byte : nombre compris entre 0 et 255)
- `h` : pour les shorts signés
- `H` : pour les shorts non-signés
- `i` : pour les entiers signés
- `I` : pour les entiers non-signés
- `l` : pour les longs signés
- `L` : pour les longs non-signés
- `f` : pour les flottants
- `d` : pour les doubles

Python

Pour utiliser les tableaux statiques, il faut faire l'import suivant

```
from array import array
```

© Achref EL MOUELHI ©

Python

Pour utiliser les tableaux statiques, il faut faire l'import suivant

```
from array import array
```

Pour déclarer un tableau

```
arr = array('i')
```

© Achref EL MOUADJIB ©

Python

Pour utiliser les tableaux statiques, il faut faire l'import suivant

```
from array import array
```

Pour déclarer un tableau

```
arr = array('i')
```

Pour déclarer un tableau avec quelques valeurs initiales

```
arr = array('i', [2, 3, 8, -5])
```

Python

Pour utiliser les tableaux statiques, il faut faire l'import suivant

```
from array import array
```

Pour déclarer un tableau

```
arr = array('i')
```

Pour déclarer un tableau avec quelques valeurs initiales

```
arr = array('i', [2, 3, 8, -5])
```

Pour afficher les caractéristiques du tableau

```
print(arr)  
# affiche array('i', [2, 3, 8, -5])
```

Python

Pour connaître le nombre d'éléments d'un tableau

```
print(len(arr))  
# affiche 4
```

© Achref EL MOUELHI ©

Python

Pour connaître le nombre d'éléments d'un tableau

```
print(len(arr))  
# affiche 4
```

Pour ajouter un élément (à la fin)

```
arr.append(10)  
  
print(arr)  
# affiche array('i', [2, 3, 8, -5, 10])
```

Python

Pour connaître le nombre d'éléments d'un tableau

```
print(len(arr))  
# affiche 4
```

Pour ajouter un élément (à la fin)

```
arr.append(10)  
  
print(arr)  
# affiche array('i', [2, 3, 8, -5, 10])
```

Pour ajouter un élément à une position donnée (ici 2)

```
arr.insert(2, 6)  
  
print(arr)  
# affiche array('i', [2, 3, 6, 8, -5])
```

Python

Pour récupérer l'indice de la première occurrence d'un élément dans le tableau, on utilise `index`. Si l'élément n'existe pas, la méthode lève une exception.

```
print(arr.index(8))  
# affiche 2
```

© Achref EL MOUL

Python

Pour récupérer l'indice de la première occurrence d'un élément dans le tableau, on utilise `index`. Si l'élément n'existe pas, la méthode lève une exception.

```
print(arr.index(8))  
# affiche 2
```

Pour supprimer un élément, on utilise `remove`. Si l'élément n'existe pas, la méthode lève une exception.

```
arr.remove(3)  
  
print(arr)  
# affiche array('i', [2, 8, -5])
```

Python

Autres opérations sur les tableaux statiques

- `count(x)` renvoi le nombre d'occurrences de `x` dans le tableau.
- `fromlist(list)` ajoute les éléments de la liste à notre tableau
- `pop([i])` supprime l'élément d'indice `i` du tableau
- `reverse()` inverse l'ordre des éléments du tableau.
- `tolist()` convertit le tableau en une liste ordinaire avec les mêmes éléments.
- ...

Python

array VS list

- `list` peut contenir des éléments de types différents mais pas `array`.
- `list` est généralement plus grande en termes d'utilisation de la mémoire pour stocker le même nombre d'éléments que `array`.
- `list` offre un large éventail de méthodes qui permettent des opérations telles que l'ajout, la suppression et la modification.

Python

typing

- Introduit dans **Python 3.5**.
- Facilitant ainsi le typage statique.
- Fournissant des outils pour spécifier les types de variables et les retours de fonction.

Python

typing : vue d'ensemble

- **Généricité** : `TypeVar` : **et** `Generic` [T]
- **Types spéciaux** : `Union`, `Any`, `Optional` [T], `NoReturn`...
- **Valeurs autorisées** : `Literal`
- ...

Python

Exemple avec `Literal`

```
from typing import Literal

def couleur_preferee(couleur: Literal['rouge', 'vert', 'bleu']):
    print(f"Ma couleur préférée est {couleur}")

couleur_preferee('rouge')
# affiche : Ma couleur préférée est rouge

couleur_preferee('jaune')
# erreur avec MyPy
```

Python

Avec `typing` (Python 3.5), il est possible d'utiliser les itérables génériques suivants

- `List[]`
- `Tuple[]`
- `Dict[]`
- `Set[]`
- `Frozenset[]`
- `Deque[]`

Python

Remarque

Depuis **Python 3.9**, il est possible d'utiliser les types built-in génériques de manière similaire aux types du module `typing`

- `list[]`
- `tuple[]`
- `dict[]`
- `set[]`
- `frozenset[]`

Python

Pour récupérer la data du jour

```
from datetime import datetime

date = datetime.now()
print(date)
# affiche 2020-06-25 21:20:02.420685
```

© Achref EL MOUËZ

Python

Pour récupérer la data du jour

```
from datetime import datetime

date = datetime.now()
print(date)
# affiche 2020-06-25 21:20:02.420685
```

`datetime.today()` VS `datetime.now()`

- `datetime.now()` : retourne la date et l'heure actuelles, avec la possibilité d'ajuster le résultat au fuseau horaire spécifié. Si aucun fuseau horaire n'est fourni, elle retourne l'heure locale.
- `datetime.today()` : retourne la date et l'heure actuelles, sans la possibilité d'ajuster le résultat au fuseau horaire spécifié.

Python

Pour extraire les différentes parties de la date

```
from datetime import datetime

date = datetime.now()

annee = date.year
mois = date.month
jour = date.day
heure = date.hour
minutes = date.minute

print(f"Année: {annee}")
print(f"Mois: {mois}")
print(f"Jour: {jour}")
print(f"Heure: {heure}")
print(f"Minutes: {minutes}")
```

Python

Pour récupérer le nom du mois de la date actuelle en toute lettre

```
print(date.strftime('%B'))  
# affiche June
```

© Achref EL MOUELHI ©

Python

Pour récupérer le nom du mois de la date actuelle en toute lettre

```
print(date.strftime('%B'))  
# affiche June
```

Pour récupérer le nom du mois de la date actuelle en toute lettre en français

```
import datetime  
import locale  
  
locale.setlocale(locale.LC_TIME, 'fr_FR')  
date = datetime.datetime.now()  
print(date.strftime('%B'))  
# affiche juin
```

Python

Quelques autres indices pour les dates

- `d` : le jour du mois (de 01 à 31)
- `a` : une représentation textuelle du jour (trois lettres)
- `A` : une représentation textuelle complète du jour
- `w` : une représentation numérique du jour (0 pour dimanche, 6 pour samedi)
- `j` : le jour de l'année (de 001 à 366)
- `B` : une représentation textuelle complète du mois (January à December)
- `m` : une représentation numérique du mois (de 01 à 12)
- `b` : une représentation textuelle courte du mois (trois lettres)
- `Y` : une représentation numérique de l'année (4 chiffres)
- `y` : une représentation numérique de l'année (2 chiffres)

Quelques autres indices pour les heures

- **p** : AM ou PM en majuscule
- **I** : format d'heure de 00 à 12
- **H** : format d'heure de 00 à 23
- **M** : minutes avec un zéro au début (de 00 à 59)
- **S** : secondes avec un zéro au début (de 00 à 59)
- **f** : microsecondes
- **Z** : fuseau horaire (exemples : UTC, GMT, Atlantic/Azores)
- **T** : abréviations du fuseau horaire (exemples : EST, MDT)

Python

Pour afficher une date sous un format précis, on utilise les options précédentes

```
print (date1.strftime ('%Y/%m/%d %H:%M:%S'))
```

© Achref MOUJAHID

Python

Pour ajouter ou soustraire un certain nombre de jours (ou autre d'une `datetime`)

```
from datetime import datetime, timedelta

# Créer une date initiale
date_initiale = datetime(2024, 2, 4)

# Ajouter 10 jours
date_plus_10_jours = date_initiale + timedelta(days=10)

print(f"Date initiale: {date_initiale.date()}")
# affiche Date initiale: 2024-02-04

print(f"Après ajout de 10 jours: {date_plus_10_jours.date()}")
#affiche Après ajout de 10 jours: 2024-02-14

date_moins_5_jours = date_initiale - timedelta(days=5)

print(f"Après soustraction de 5 jours: {date_moins_5_jours.date()}")
# affiche Après soustraction de 5 jours: 2024-01-30
```

Python

Pour comparer les dates, on peut utiliser les opérateurs de comparaison

```
from datetime import datetime

date1 = datetime(2023, 2, 4)
date2 = datetime(2024, 3, 1)

print(f"Date1 est avant Date2: {date1 < date2}")
# affiche Date1 est avant Date2 : True

print(f"Date1 est après Date2: {date1 > date2}")
# affiche Date1 est après Date2 : False

print(f"Date1 est la même que Date2: {date1 == date2}")
# affiche Date1 est la même que Date2 : False
```

Python

Pour calculer la différence entre deux dates en jours

```
from datetime import datetime

date1 = datetime(2023, 2, 4)
date2 = datetime(2024, 3, 1)

difference = date2 - date1
print(f"Différence en jours: {difference.days}")
# affiche Différence en jours: 391
```

Python

Pour calculer la différence entre deux dates en jours

```
from datetime import datetime, timezone, timedelta

paris_time = datetime.now()
print("Heure actuelle à Paris :", paris_time)
# affiche Heure actuelle à Paris : 2020-02-25 21:20:02.420685
```

© Achref EL MOUËLTI

Python

Pour calculer la différence entre deux dates en jours

```
from datetime import datetime, timezone, timedelta

paris_time = datetime.now()
print("Heure actuelle à Paris :", paris_time)
# affiche Heure actuelle à Paris : 2020-02-25 21:20:02.420685
```

Construisons le fuseau horaire de New York (5 heures de moins par rapport à UTC)

```
ny_tz = timezone(timedelta(hours=-5))
```

Python

Pour calculer la différence entre deux dates en jours

```
from datetime import datetime, timezone, timedelta

paris_time = datetime.now()
print("Heure actuelle à Paris :", paris_time)
# affiche Heure actuelle à Paris : 2020-02-25 21:20:02.420685
```

Construisons le fuseau horaire de New York (5 heures de moins par rapport à UTC)

```
ny_tz = timezone(timedelta(hours=-5))
```

Ensuite, nous pouvons obtenir l'heure de New York à partir de l'heure de Paris)

```
ny_time = paris_time.astimezone(ny_tz)
print("Heure actuelle à New York :", ny_time)
# affiche Heure actuelle à New York : 2020-02-25 15:20:02.420685-05:00
```

Python

pytz

- Librairie externe, populaire avant l'arrivée de `zoneinfo`.
- Utilisé souvent avec le module `datetime` pour une meilleure gestion des fuseaux horaires.
- Supporte les règles de changement d'heure saisonnier (**DST, Daylight Saving Time**)

© Actif

Python

pytz

- Librairie externe, populaire avant l'arrivée de `zoneinfo`.
- Utilisé souvent avec le module `datetime` pour une meilleure gestion des fuseaux horaires.
- Supporte les règles de changement d'heure saisonnier (**DST, Daylight Saving Time**)

Pour installer `pytz`

```
pip install pytz
```

Python

Pour récupérer la data du jour selon le fuseau horaire de New York

```
timezone = pytz.timezone('America/New_York')
heure_ny = datetime.now(timezone)

print(heure_ny)
# affiche 2020-06-25 14:23:49.049765-04:00
```

© Achref EL M...

Python

Pour récupérer la data du jour selon le fuseau horaire de New York

```
timezone = pytz.timezone('America/New_York')
heure_ny = datetime.now(timezone)

print(heure_ny)
# affiche 2020-06-25 14:23:49.049765-04:00
```

Pour récupérer le fuseau horaire d'un datetime

```
tz = heure_ny.tzinfo

print(tz)
# affiche America/New_York
```

Remarque

`tzinfo` peut être `None` si l'objet `datetime` a été créé sans spécification de fuseau horaire.

Python

zoneinfo

- Intégré dans **Python** depuis la version 3.9
- Devenu la méthode standard pour les fuseaux horaires en **Python**
- Supporte les règles de changement d'heure saisonnier (**DST**, **Daylight Saving Time**)

Python

Exemple

```
from datetime import datetime
from zoneinfo import ZoneInfo

ny_tz = ZoneInfo("America/New_York")

time_ny = datetime.now(ny_tz)
print("Heure actuelle à New York :", time_ny)

la_tz = ZoneInfo("America/Los_Angeles")
time_la = time_ny.astimezone(la_tz)
print("Heure actuelle à Los Angeles :", time_la)
```

Python

Propriété	<code>datetime.timezone</code>	<code>pytz</code>	<code>zoneinfo</code>
Intégré dans Python	Oui	Non, externe	Oui (Python 3.9+)
Fuseaux horaires fixes	Oui	Oui	Oui
Gestion des changements DST	Non	Oui	Oui

Python

Pour résumer

- Pour des fuseaux horaires fixes (sans **DST**), `datetime.timezone` suffit.
- Pour des applications nécessitant des fuseaux horaires précis avec **DST**, `zoneinfo` est le meilleur choix car il est intégré.
- Si vous utilisez une version antérieure à **Python 3.9**, **pytz** reste une bonne option.

Python

Liste des fuseaux horaires

https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

© Achrel & HI ©

Python

Commençons par importer `time`

```
from time import *
```

© Achref EL MOUËLHI

Python

Commençons par importer `time`

```
from time import *
```

Pour obtenir le timestamp actuel

```
timestamp = time()
```

```
print("Timestamp actuel :", timestamp)
```

```
Timestamp actuel : 1707979755.572501
```

Python

Pour obtenir le timestamp à partir d'un `datetime`

```
from datetime import datetime

now = datetime.now()
timestamp = datetime.timestamp(now)
print("Timestamp actuel:", timestamp)
# affiche Timestamp actuel: 1707979755.572501
```

© Achref EL MOU

Python

Pour obtenir le timestamp à partir d'un `datetime`

```
from datetime import datetime

now = datetime.now()
timestamp = datetime.timestamp(now)
print("Timestamp actuel:", timestamp)
# affiche Timestamp actuel: 1707979755.572501
```

Pour obtenir un `datetime` à partir d'un timestamp

```
from datetime import datetime

timestamp = 1644921600

date = datetime.fromtimestamp(timestamp)

print("datetime :", date)
# affiche datetime : 2022-02-15 11:40:00
```

Quelques fonctions prédéfinies sur les collections utilisant callback et Lambda

- `map(callback, iterable)` **[Built-in function]**
- `filter(callback, iterable)` **[Built-in function]**
- `reduce(callback, iterable, [initial])`

Python

Exemple avec `map`

```
tab = [2, 3, 8, 5]

def carre(x):
    return x ** 2

result = list(map(carre, tab))

print(result)
# affiche [4, 9, 64, 25]
```

© Achille

Python

Exemple avec `map`

```
tab = [2, 3, 8, 5]

def carre(x):
    return x ** 2

result = list(map(carre, tab))

print(result)
# affiche [4, 9, 64, 25]
```

Ou en utilisant une fonction Lambda

```
result = list(map(lambda x: x ** 2, tab))

print(result)
# affiche [4, 9, 64, 25]
```

Python

Exemple avec `filter`

```
tab = [2, 3, 8, 5]

def est_pair(x):
    return x % 2 == 0

result = list(filter(est_pair, tab))

print(result)
# affiche [2, 8]
```

© Achille

Python

Exemple avec `filter`

```
tab = [2, 3, 8, 5]

def est_pair(x):
    return x % 2 == 0

result = list(filter(est_pair, tab))

print(result)
# affiche [2, 8]
```

Ou en utilisant une fonction Lambda

```
result = list(filter(lambda x: x % 2 == 0, tab))

print(result)
# affiche [2, 8]
```

Python

`map`, `filter` **et** `reduce` **peuvent être appliqués sur les tableaux statiques**

```
from array import array

arr = array('i', [2, 3, 8, -5])

print(array('i', (map(lambda elt: elt + 2, arr))))
# affiche array('i', [4, 5, 10, -3])
```

Python

Exemple avec `reduce`

```
from functools import reduce
```

```
tab = [2, 3, 8, 5]
```

```
def somme(x, y):  
    return x + y
```

```
result = reduce(somme, tab)  
print(result)  
# affiche 18
```

Python

Exemple avec `reduce`

```
from functools import reduce
```

```
tab = [2, 3, 8, 5]
```

```
def somme(x, y):  
    return x + y
```

```
result = reduce(somme, tab)  
print(result)  
# affiche 18
```

Ou en utilisant une fonction Lambda

```
result = reduce(lambda x, y: x + y, tab)
```

```
print(result)  
# affiche 18
```

TypeScript

Remarques

- Le premier paramètre de `reduce` correspond à une fonction callback ou Lambda
 - Le premier paramètre de la fonction callback ou Lambda correspond au résultat
 - Le deuxième correspond à l'élément du tableau de l'itération courante
 - Le premier paramètre est initialisé par la valeur du premier élément du tableau
- Le deuxième correspond à la liste
- On peut ajouter un troisième paramètre qui correspondra à la valeur initiale du premier paramètre de la fonction callback ou Lambda

Python

Exemple avec `reduce` utilisant un troisième paramètre

```
from functools import reduce
```

```
tab = [2, 3, 8, 5]
```

```
result = reduce(lambda x, y: x + y, tab, 5)
```

```
print(result)
```

```
# affiche 23
```

Python

Pour enchaîner les différentes méthodes précédentes, on fait des imbrications

```
from functools import reduce

tab = [2, 3, 8, 5]
result = (reduce(lambda x, y: x + y, filter(lambda x
      : x % 2 == 0, map(lambda x: x ** 2, tab))))

print(result)
# affiche 68
```

Python

Remarques

- Le code précédent permettait d'imbriquer les méthodes.
- Il ne permettait pas le chaînage comme en **Java**, **C#** et **JavaScript** (Approche fonctionnelle)
- Pour pouvoir enchaîner les méthodes précédentes comme en **Java**, **JavaScript** et **C#**, on peut utiliser **PyFunctional** (anciennement appelé **ScalaFunctional**)

Python

Remarques

- Le code précédent permettait d'imbriquer les méthodes.
- Il ne permettait pas le chaînage comme en **Java**, **C#** et **JavaScript** (Approche fonctionnelle)
- Pour pouvoir enchaîner les méthodes précédentes comme en **Java**, **JavaScript** et **C#**, on peut utiliser **PyFunctional** (anciennement appelé **ScalaFunctional**)

Pour installer `PyFunctional`

```
pip install PyFunctional
```

Python

Ensuite, nous pourrons enchaîner ces méthodes de la manière suivante

```
from functional import seq

tab = [2, 3, 8, 5]

result = seq(tab)\
    .map(lambda x: x ** 2)\
    .filter(lambda x: x % 2 == 0)\
    .reduce(lambda x, y: x + y)

print(result)
# affiche 68
```

Python

Ou avec une syntaxe Linq (de Microsoft)

```
from functional import seq

tab = [2, 3, 8, 5]

result = seq(tab)\
    .select(lambda x: x ** 2)\
    .where(lambda x: x % 2 == 0)\
    .reduce(lambda x, y: x + y)

print(result)
# affiche 68
```

Python

Étant donné le tuple suivant :

```
marques = ("peugeot", "ford", "toyota")
```

Exercice

En utilisant `map`, `filter` et `reduce`, calculer le nombre de caractères total des marques ayant un nombre pair de caractères.

Python

Solution

```
from functional import seq

marques = ("peugeot", "ford", "toyota")

result = seq(marques)\
    .map(lambda x: len(x))\
    .filter(lambda x: x % 2 == 0)\
    .reduce(lambda x, y: x + y)

print(result)
# affiche 10
```

Python

Autres fonctions de `functional`

- `find(func)`
- `flat_map(func)`
- `drop(n)`
- `drop_right(n)`
- `drop_while(func)`
- `for_each(func)`
- `count(func)`
- ...

Python

Quelques autres exemples

```
from functional import seq

tab = [2, 3, 8, 5]

print(seq(tab).find(lambda v: v % 2 == 0))
# affiche 2

print(seq(tab).count(lambda v: v % 2 == 0))
# affiche 2
```

Python

Fonctions de quantification

- `take(n)`
- `take_while(func)`
- `any()`
- `exists(func)`
- `all()`
- `for_all(func)`
- ...

Python

Exemples avec les opérateurs de quantification

```
from functional import seq
```

```
tab = [2, 3, 8, 5]
```

```
print(seq(tab).all())
```

```
# affiche True
```

```
print(seq(tab).any())
```

```
# affiche True
```

```
print(seq(tab).exists(lambda v: v % 2 == 0))
```

```
# affiche True
```

```
print(seq(tab).for_all(lambda v: v % 2 == 0))
```

```
# affiche False
```

Python

Fonctions d'agrégation de `functional`

- `min()`
- `min_by(func)`
- `max()`
- `max_by(func)`
- `average`, `sum`, `product`
- `distinct()`
- `distinct_by(func)`
- ...

Python

Built-in functions vs `Functional`

- L'utilisation de `Functional` peut être plus claire pour ceux qui préfèrent la programmation fonctionnelle
- Pour de petites à moyennes collections, la différence de performance entre ces deux approches est généralement négligeable.
- `Functional` pourrait ajouter un léger surcoût en raison des abstractions supplémentaires : un surcoût souvent minime et n'est significatif que dans des scénarios de performance critique ou avec de très grands ensembles de données.

Python

Pour en apprendre plus

- <https://pypi.org/project/ScalaFunctional/>
- <https://docs.pyfunctional.pedro.ai/en/latest/functional.html#module-functional.pipeline>

Python

collections

Le module `collections` offre plusieurs classes utiles pour manipuler les itérables

- `namedtuple` : permet de créer des tuples avec des noms de champs.
- `deque` : implémente une double file (queue) qui prend en charge l'ajout et la suppression efficaces d'éléments aux deux extrémités de la file.
- `Counter` : utilisé pour compter les occurrences d'éléments dans une séquence.
- ...

Python

Exemple d'utilisation pour compter les occurrences d'éléments d'une liste

```
from collections import Counter

liste = ['a', 'b', 'c', 'a', 'b', 'a']
compteur = Counter(liste)

print(compteur)
# affiche Counter({'a': 3, 'b': 2, 'c': 1})
```

Python

Un compteur peut être construit à partir d'un dictionnaire

```
dictionnaire = {'a': 3, 'b': 2, 'c': 1}
compteur = Counter(dictionnaire)
```

© Achref EL MOUELHI ©

Python

Un compteur peut être construit à partir d'un dictionnaire

```
dictionnaire = {'a': 3, 'b': 2, 'c': 1}
compteur = Counter(dictionnaire)
```

`Counter` fournit également des fonctionnalités comme la soustraction, l'addition, l'intersection et l'union de compteurs

```
compteur1 = Counter({'a': 3, 'b': 2, 'c': 1})
compteur2 = Counter({'a': 1, 'b': 2, 'd': 1})

somme = compteur1 + compteur2

print(somme)
# Counter({'a': 4, 'b': 4, 'c': 1, 'd': 1})

difference = compteur1 - compteur2

print(difference)
# Counter({'a': 2, 'c': 1})
```

Python

Fonctionnalités du module `os`

- Navigation dans le système de fichiers : `getcwd`, `chdir`, `listdir`...
- Gestion de fichiers et de répertoires : `mkdir`, `rmdir`, `rename`...
- Manipulation de chemins : `path.dirname`, `path.basename`...
- Exécution de commandes système : `system(...)`
- Travail avec des variables d'environnement : `getenv`...
- Gestion des permissions et des identifiants : `chmod`, `chown`...

Python

Fonctionnalités du module `os`

- Navigation dans le système de fichiers : `getcwd`, `chdir`, `listdir`...
- Gestion de fichiers et de répertoires : `mkdir`, `rmdir`, `rename`...
- Manipulation de chemins : `path.dirname`, `path.basename`...
- Exécution de commandes système : `system(...)`
- Travail avec des variables d'environnement : `getenv`...
- Gestion des permissions et des identifiants : `chmod`, `chown`...

Commençons par importer le module

```
import os
```

Python

Pour récupérer le nom du répertoire de travail (CWD : Current Working Directory) du fichier utilisé

```
print(os.getcwd())  
# affiche C:\Users\user\cours-modules
```

© Achref EL MOUELHI ©

Python

Pour récupérer le nom du répertoire de travail (CWD : Current Working Directory) du fichier utilisé

```
print(os.getcwd())  
# affiche C:\Users\user\cours-modules
```

Pour récupérer les fichiers d'un répertoire passé en paramètre (sinon le répertoire courant)

```
arr = os.listdir()  
print(arr)  
# affiche ['.idea', 'main.py', 'package', 'venv', '__pycache__']
```

Python

Pour récupérer le nom du répertoire de travail (CWD : Current Working Directory) du fichier utilisé

```
print(os.getcwd())  
# affiche C:\Users\user\cours-modules
```

Pour récupérer les fichiers d'un répertoire passé en paramètre (sinon le répertoire courant)

```
arr = os.listdir()  
print(arr)  
# affiche ['.idea', 'main.py', 'package', 'venv', '__pycache__']
```

Pour changer de répertoire, on utilise `chdir()`

```
os.chdir('package')  
print(os.getcwd())  
# affiche C:\Users\user\cours-modules\package
```

Python

Pour créer un répertoire

```
os.mkdir('a')
```

© Achref EL MOUËLHA

Python

Pour créer un répertoire

```
os.mkdir('a')
```

`mkdir`

- lance une exception si le répertoire existe
- ne permet pas de créer une arborescence de répertoires

Python

Pour créer une arborescence de répertoires

```
os.makedirs('b/c/d')
```

© Achref EL MOUELHI ©

Python

Pour créer une arborescence de répertoires

```
os.makedirs('b/c/d')
```

Remarque

`makedirs` lance une exception si le répertoire existe déjà

Python

Pour créer une arborescence de répertoires

```
os.makedirs('b/c/d')
```

Remarque

`makedirs` lance une exception si le répertoire existe déjà

Pour créer récursivement des répertoires et ne pas lever d'exception si le répertoire existe déjà

```
os.makedirs('b/c/d', exist_ok=True)
```

Python

Pour supprimer un répertoire vide

```
os.rmdir('a')
```

© Achref EL MOUËLHI ©

Python

Pour supprimer un répertoire vide

```
os.rmdir('a')
```

`rmdir`

- lance une exception si le répertoire n'existe pas
- ne permet pas de supprimer un répertoire non vide
- ne permet pas de supprimer une arborescence de répertoires

Python

Pour supprimer un répertoire et tous ses sous-répertoires vides de manière récursive

```
os.removedirs('b\c\d')
```

© Achref EL MOUËL

Python

Pour supprimer un répertoire et tous ses sous-répertoires vides de manière récursive

```
os.removedirs('b\c\d')
```

Remarques

- `removedirs` ne permet pas de supprimer un répertoire non vide
- Pour supprimer un répertoire non-vidé, il faut utiliser module `shutil`

Python

Pour renommer un répertoire

```
os.rename('package', 'paquet')
```

© Achref EL MOULI

Python

Pour renommer un répertoire

```
os.rename('package', 'paquet')
```

Pour supprimer un fichier

```
os.remove('file.py')
```

Python

```
os.system()
```

`os.system()` prend comme paramètre une commande qu'on peut exécuter dans un invite de commande (terminal).

Python

Pour vider l'écran (sous Windows)

```
os.system('cls')
```

© Achref EL MOUELHI ©

Python

Pour vider l'écran (sous Windows)

```
os.system('cls')
```

Pour vider l'écran (sous Linux)

```
os.system('clear')
```

© Achref MOUELHI ©

Python

Pour vider l'écran (sous Windows)

```
os.system('cls')
```

Pour vider l'écran (sous Linux)

```
os.system('clear')
```

Pour récupérer la date système

```
os.system('date')
```

Python

Pour construire un chemin relatif avec une gestion correcte des séparateurs.

```
chemin = os.path.join('paquet', 'subpackage', 'fonctions.py')  
  
print (chemin)  
# affiche paquet\subpackage\fonctions.py
```

Python

Pour avoir des informations sur un chemin, on utilise `os.path()`

```
print(os.path.dirname(chemin))
```

```
# affiche paquet\subpackage
```

```
print(os.path.basename(chemin))
```

```
# affiche fonctions.py
```

```
print(os.path.exists(chemin))
```

```
# affiche True
```

```
print(os.path.isfile(chemin))
```

```
#affiche True
```

```
print(os.path.isdir(chemin))
```

```
#affiche False
```

Python

Exercice

- Affichez le contenu du répertoire courant
- Pour chaque élément, affichez :
 - s'il est répertoire
 - s'il est fichier
 - son chemin absolu

Python

Pour afficher tous les dossiers du répertoire de travail, on utilise `walk` qui retourne à chaque itération le chemin vers le répertoire parcouru, ses sous-répertoires et ses fichiers

```
for root, dirs, files in os.walk(os.getcwd()):  
    for file in files:  
        print(os.path.join(root, file))
```

Python

glob

- fonction de recherche selon un motif
- similaire à `walk` du module `os`

© Achref EL M...

Python

glob

- fonction de recherche selon un motif
- similaire à `walk` du module `os`

Commençons par importer `glob`

```
from glob import *
```

Python

Pour afficher le contenu du répertoire de travail (la racine du projet)

```
for elt in glob("*"):  
    print(elt)
```

© Achref EL MOUELHI ©

Python

Pour afficher le contenu du répertoire de travail (la racine du projet)

```
for elt in glob("*"):  
    print(elt)
```

Pour afficher le contenu des sous répertoires définis dans le répertoire de travail (la racine du projet)

```
for elt in glob("**/*"):  
    print(elt)
```

Python

Pour afficher le contenu du répertoire de travail (la racine du projet)

```
for elt in glob("*"):  
    print(elt)
```

Pour afficher le contenu des sous répertoires définis dans le répertoire de travail (la racine du projet)

```
for elt in glob("**/*"):  
    print(elt)
```

Depuis Python 3.5, glob permet d'effectuer une recherche récursive

```
for elt in glob("**/*", recursive=True):  
    print(elt)
```

Python

Pour afficher les fichiers ayant une extension `.py` situés dans les sous répertoires définis dans le répertoire de travail (la racine du projet)

```
for elt in glob("**/*.py"):  
    print(elt)
```

© Achref EL MOU

Python

Pour afficher les fichiers ayant une extension `.py` situés dans les sous répertoires définis dans le répertoire de travail (la racine du projet)

```
for elt in glob("**/*.py"):  
    print(elt)
```

Pour une recherche en profondeur des fichiers ayant une extension `.py` dans le répertoire de travail (la racine du projet)

```
for elt in glob("**/*.py", recursive=True):  
    print(elt)
```

Python

`os.walk` VS `glob`

- `os.walk` est principalement conçu pour parcourir récursivement une arborescence de répertoires.
- `os.walk` n'effectue pas de filtrage basé sur des motifs.
- `glob.glob` n'est pas récursif par défaut mais permet d'effectuer une recherche selon un motif.

Python

shutil : shell utilities

propose des opérations de haut niveau

- `rmtree()` pour supprimer un répertoire non-vide
- `copyfile()` pour copier un fichier
- `copytree()` pour copier un répertoire non-vide
- `move()` pour déplacer un répertoire non-vide
- ...

Python

shutil : shell utilities

propose des opérations de haut niveau

- `rmtree()` pour supprimer un répertoire non-vide
- `copyfile()` pour copier un fichier
- `copytree()` pour copier un répertoire non-vide
- `move()` pour déplacer un répertoire non-vide
- ...

Commençons par importer `shutil`

```
import shutil
```

Python

Pour copier le répertoire non vide `paquet` **dans** `package` **(qui sera créé)**

```
shutil.copytree('paquet', 'package')
```

© Achref EL MOUELHI ©

Python

Pour copier le répertoire non vide `paquet` **dans** `package` **(qui sera créé)**

```
shutil.copytree('paquet', 'package')
```

Pour supprimer le répertoire non vide et toute l'arborescence

```
shutil.rmtree('paquet')
```

Python

Pour copier le répertoire non vide `paquet` **dans** `package` **(qui sera créé)**

```
shutil.copytree('paquet', 'package')
```

Pour supprimer le répertoire non vide et toute l'arborescence

```
shutil.rmtree('paquet')
```

Pour déplacer `package` **dans** `paquet` **(qui sera recréé, package sera supprimé)**

```
shutil.move('package', 'paquet')
```

Exercice

Écrire un script **Python** qui permet de :

- 1 afficher le chemin du répertoire de travail
- 2 créer un répertoire `a`
- 3 se déplacer dans `a`
- 4 afficher le chemin du répertoire de travail
- 5 dans `a` créer trois fichiers `f1.txt`, `f2.txt` et `f3.txt`
- 6 dans `a` créer deux répertoires `b` et `c`
- 7 supprimer `f2.txt`
- 8 renommer `f3.txt` en `f2.txt`
- 9 se déplacer dans le répertoire parent (racine du projet)
- 10 afficher le chemin du répertoire de travail
- 11 lister le contenu de `a` en précisant pour chaque élément s'il est fichier ou répertoire

Python

Correction

```
import os, glob

# question 1
print(os.getcwd())
# question 2
os.system('mkdir a')
# question 3
os.chdir('a')
# question 4
print(os.getcwd())
# question 5
os.system('copy NUL f1.txt')
os.system('copy NUL f2.txt')
os.system('copy NUL f3.txt')
# question 6
os.mkdir('b')
os.system('mkdir c')
# question 7
os.remove('f2.txt')
# question 8
os.rename('f3.txt', 'f2.txt')
# question 9
os.chdir('..')
# question 10
print(os.getcwd())
# question 11
for elt in glob.glob("a/*", recursive=True):
    print(elt, 'file' if os.path.isfile(elt) else 'directory')
```

Pour supprimer le répertoire `a` s'il existe à chaque exécution du projet on utilise la méthode `rmtree()` de `shutil` (qui permet de supprimer un répertoire non vide)

```
import os, glob, shutil

if os.path.exists('a'):
    shutil.rmtree('a', ignore_errors=True)
# question 1
print(os.getcwd())
# question 2
os.system('mkdir a')
# question 3
os.chdir('a')
# question 4
print(os.getcwd())
# question 5
os.system('copy NUL f1.txt')
os.system('copy NUL f2.txt')
os.system('copy NUL f3.txt')
# question 6
os.mkdir('b')
os.system('mkdir c')
# question 7
os.remove('f2.txt')
# question 8
os.rename('f3.txt', 'f2.txt')
# question 9
os.chdir('..')
# question 10
print(os.getcwd())
# question 11
for elt in glob.glob("a/*", recursive=True):
    print(elt, 'file' if os.path.isfile(elt) else 'directory')
```

Python

sys

module fournissant l'accès à certaines variables et fonctions qui ont une forte interaction avec l'interpréteur **Python**

© Achref EL M...

Python

sys

module fournissant l'accès à certaines variables et fonctions qui ont une forte interaction avec l'interpréteur **Python**

Commençons par importer `sys`

```
import sys
```

Python

Pour récupérer les paramètres envoyés au lancement du programme

```
print(sys.argv)
```

© Achref EL MOUELHI ©

Python

Pour récupérer les paramètres envoyés au lancement du programme

```
print(sys.argv)
```

Pour lancer le programme

```
python main.py 2 5 8
```

Python

Pour récupérer les paramètres envoyés au lancement du programme

```
print(sys.argv)
```

Pour lancer le programme

```
python main.py 2 5 8
```

Résultat

```
['main.py', '2', '5', '8']
```

Python

`sys` peut aussi nous permettre de quitter le programme

```
if len(sys.argv) < 2:  
    sys.exit("Erreur détectée, arrêt du programme.")  
  
print(sys.argv)
```

© Achref EL MOUELHI

Python

`sys` peut aussi nous permettre de quitter le programme

```
if len(sys.argv) < 2:  
    sys.exit("Erreur détectée, arrêt du programme.")  
  
print(sys.argv)
```

Pour lancer le programme

```
python main.py
```

Python

`sys` peut aussi nous permettre de quitter le programme

```
if len(sys.argv) < 2:  
    sys.exit("Erreur détectée, arrêt du programme.")  
  
print(sys.argv)
```

Pour lancer le programme

```
python main.py
```

Résultat

```
Erreur détectée, arrêt du programme.
```

Python

Pour récupérer la version de Python utilisée

```
print(sys.version)
# affiche 3.11.7 (tags/v3.11.7:fa7a6f2, Dec 4 2023, 19:24:49) [MSC v
.1937 64 bit (AMD64)]
```

Python

```
importlib
```

module permettant de réaliser une importation dynamique

© Achref EL MOUELHI ©

Python

```
importlib
```

module permettant de réaliser une importation dynamique

Considérons le fichier `fr.py` ayant le contenu suivant

```
def salutation():  
    print('salut')
```

Python

```
importlib
```

module permettant de réaliser une importation dynamique

Considérons le fichier `fr.py` ayant le contenu suivant

```
def salutation():  
    print('salut')
```

Et `en.py` avec le contenu suivant

```
def salutation():  
    print('hello')
```

Python

Importons `importlib` et réalisons l'import dynamique suivant

```
import importlib
import locale

language = "fr" if locale.getlocale()[0] == 'fr_FR' else "en"
module = importlib.import_module(language)

module.salutation()
```

Python

Considérons la liste suivante

```
liste1 = [1, 2, 3, 4]
```

© Achref EL MOUELHI ©

Python

Considérons la liste suivante

```
liste1 = [1, 2, 3, 4]
```

Créons `liste2` **à partir de** `liste1`

```
liste2 = liste1
```

© Achret L. JOUELHI ©

Python

Considérons la liste suivante

```
liste1 = [1, 2, 3, 4]
```

Créons `liste2` à partir de `liste1`

```
liste2 = liste1
```

Modifier `liste1` ⇒ modifier `liste2`

```
liste1[1] = 10  
print(liste2)  
# affiche [1, 10, 3, 4]
```

Python

Pour résoudre le problème précédent , nous pouvons reconstruire la liste

```
liste2 = [ *liste1 ]
```

© Achref EL MOUELHI ©

Python

Pour résoudre le problème précédent , nous pouvons reconstruire la liste

```
liste2 = [ *liste1 ]
```

Modifier liste1 \nRightarrow **modifier** liste2

```
liste1[1] = 10
```

```
print(liste1)
```

```
# affiche [1, 10, 3, 4]
```

```
print(liste2)
```

```
# affiche [1, 2, 3, 4]
```

Et si la liste était plus complexe ?

```
liste3 = [[1, 2], [3, 4]]
```

© Achref EL MOUELHI ©

Et si la liste était plus complexe ?

```
liste3 = [[1, 2], [3, 4]]
```

La solution précédent ne suffira pas

```
liste4 = [ *liste3 ]  
liste3[0][0] = 10  
  
print(liste3)  
# affiche [[10, 2], [3, 4]]  
  
print(liste4)  
# affiche [[10, 2], [3, 4]]
```

Et si la liste était plus complexe ?

```
liste3 = [[1, 2], [3, 4]]
```

La solution précédent ne suffira pas

```
liste4 = [ *liste3 ]
liste3[0][0] = 10

print(liste3)
# affiche [[10, 2], [3, 4]]

print(liste4)
# affiche [[10, 2], [3, 4]]
```

Solution

Utiliser le package `copy`.

Python

Commençons par importer `copy`

```
import copy
```

© Achref EL MOUELHI ©

Python

Commençons par importer `copy`

```
import copy
```

Utilisons `deepcopy` pour cloner la liste

```
liste4 = copy.deepcopy(liste3)
```

© Achref EL MOUADJIDI ©

Python

Commençons par importer `copy`

```
import copy
```

Utilisons `deepcopy` pour cloner la liste

```
liste4 = copy.deepcopy(liste3)
```

Modifier `liste3` \neq modifier `liste4`

```
liste3[0][0] = 10
```

```
print(liste3)
```

```
# affiche [[10, 2], [3, 4]]
```

```
print(liste4)
```

```
# affiche [[1, 2], [3, 4]]
```

Python

zlib

- permet la compression et la décompression de données
- utilise l'algorithme **DEFLATE**
 - algorithme utilisé par des formats comme **gzip**, **ZIP** et **PNG**).
 - combine **LZ77** (pour la suppression des redondances) et **Huffman** (pour l'optimisation des symboles) pour une meilleure compression.
- permet un débit rapide pour la compression et la décompression.

Python

Considérons le fichier `fr.py` ayant le contenu suivant

```
import zlib

data = b"Hello, world! Hello, world!"
compressed = zlib.compress(data)
decompressed = zlib.decompress(compressed)

print("Taille originale :", len(data))
# affiche 27

print("Taille compressée :", len(compressed))
# affiche 25

print("Données décompressées :", decompressed)
# affiche Hello, world! Hello, world!
```

Python

`keyword`

module permettant de vérifier si un mot donné est un mot-clé réservé en **Python**.

© Achref EL MOUELHI ©

Python

keyword

module permettant de vérifier si un mot donné est un mot-clé réservé en **Python**.

Commençons par importer `keyword`

```
import keyword
```

© Achref EL BOUJELHI ©

Python

keyword

module permettant de vérifier si un mot donné est un mot-clé réservé en **Python**.

Commençons par importer `keyword`

```
import keyword
```

Pour tester si un mot donné est un mot-clé du langage

```
print(keyword.iskeyword("if"))  
# affiche True  
  
print(keyword.iskeyword("hello"))  
# affiche False
```

Python

Pour afficher la liste des 35 mots-clés et qui ne pourront pas être utilisés comme identifiant (nom de variables ou de fonctions)

```
print (keyword.kwlist)
# affiche ['False', 'None', 'True', 'and', 'as', 'assert', '
  async', 'await', 'break', 'class', 'continue', 'def', 'del',
  'elif', 'else', 'except', 'finally', 'for', 'from', 'global
  ', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not',
  'or', 'pass', 'raise', 'return', 'try', 'while', 'with', '
  yield']

print (len(keyword.kwlist))
# affiche 35
```

Python

Conventions

- Éviter `import *`
- Une documentation par module à placer au tout début du fichier (avant les `import`)
- Les `imports` au tout début du fichier après la documentation
- Un `import` par ligne
- Placer les `imports` dans un bloc `try ... except` pour capturer les exceptions de type `ImportError`

© ADP

Python

Conventions

- Éviter `import *`
- Une documentation par module à placer au tout début du fichier (avant les `import`)
- Les `imports` au tout début du fichier après la documentation
- Un `import` par ligne
- Placer les `imports` dans un bloc `try ... except` pour capturer les exceptions de type `ImportError`

Extension VSC

Flake8 : permet de vérifier si les bonnes pratiques sont respectées, signaler les imports inutiles...