corelibs

Release 1.0.0

Michel TRUONG

Jan 07, 2021

CONTENTS:

Note: Bienvenue dans la documentation de corelibs.

L'objectif de corelibs est d'agréger dans différents modules, toutes les fonctionnalités utiles pour simplifier nos travaux sans devoir tout réécrire à chaque fois...

CHAPTER

ONE

CONFIGURATIONS

Description générale

Ci-dessous sont listés l'ensemble des constantes définies et utilisées dans le package corelibs.

Ces constantes peuvent être utilisées telles que définies ou écrasées à discrétion :

- soit de manière globale, via le fichier **user_config.py**
- soit localement, dans les programmes python appelants.

L'ordre de recherche d'une constante est donc :

1. **localement**
2. **user_config.py**
3. **config.py** (le fichier de configuration de corelibs)

Dans le cas d'une constante simple, par exemple `DEFAULT_LOGS_EXTENSION = ".log"`, corelibs utilisera la version écrasée telle que redéfinie.

Dans le cas d'une constante plus complexe, de type dictionnaire, par exemple

```
DEFAULT_FIELD_STYLES = {
"asctime":  {"color":  242, "bright":  True},
"hostname":  {"color":  "magenta"},
"username":  {"color":  "yellow"},
"levelname":  {"color":  242, "bright":  True},
"name":  {"color":  "blue"},
"programname":  {"color":  "cyan"}
}
```

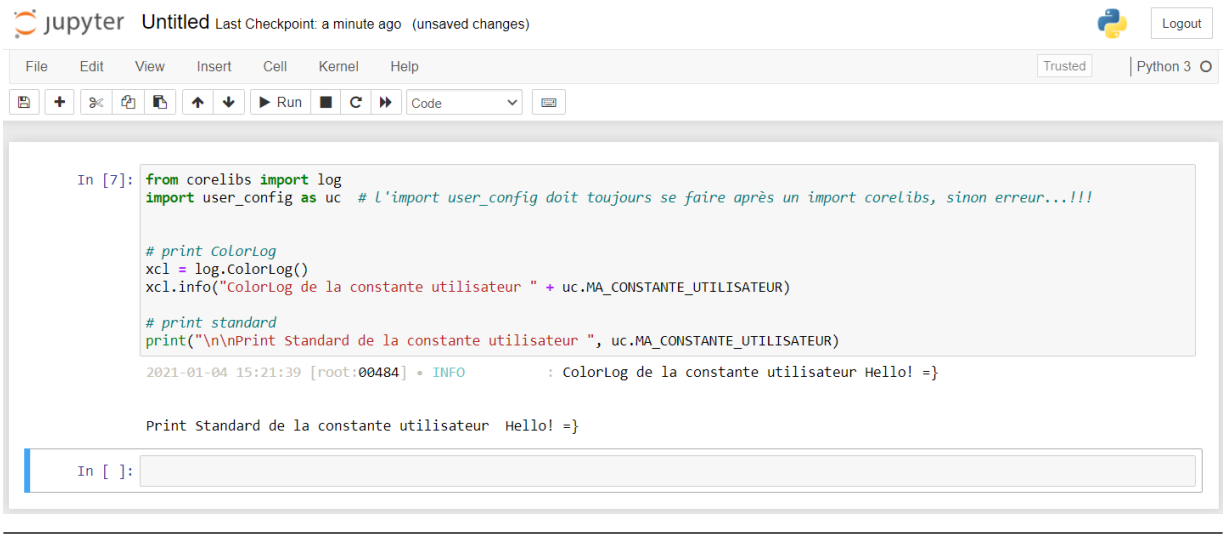
corelibs ne remplacera que les clés/valeurs redéfinies (i.e. les autres clés/valeurs seront gardées inchangées)

Note:

Le fichier **user_config.py** se trouve à l'emplacement %HOMEPATH%/.corelibs (pour les utilisateurs Windows) et ~/.corelibs (pour les utilisateurs Linux/Unix) - cf. *corelibs.lazy.open_explorer()*

Ce fichier est automatiquement inclut à l'exécution, ainsi que son dossier parent. De ce fait, il est donc possible d'y définir des constantes utilisateurs et/ou d'ajouter dans le répertoire parent des programmes python devant être inclus et exécuté globalement.

La configuration fonctionne également sur **Jupyter**



Warning:
Les écrasements sont contrôlés par des schémas de validation. Si le fichier est corrompu, il est possible de supprimer le fichier **user_config.py**. corelibs le regénèrera automatiquement.

Pour plus de détails sur les valeurs possibles, suivre la documentation officielle des packages tiers (cf. *Liens utiles* et *Dépendances*).

```
import logging as log
################################################################################
↪##################################
# Bienvenue à...
↪                                    #
#                                       _______                  ____ __
↪                                    #
#                                      / ____/___  ____  ____  / (_) /_  _____
↪                                    #
#                                     / /   / __ \/ ___/ _ \/ / / __ \/ ___/
↪                                    #
#                                    / /___/ /_/ / /  /  __/ / / /_/ (__  )
↪                                    #
#                                    \____/\____/_/   \___/_/_/_.___/____/
↪                  #################
#                                                                              
↪                 # user_config.py #
################################################################################
↪##################################
```

(continues on next page)

(continued from previous page)

```
########################################################################################
↪#####################################
# CONFIGURATION UTILISATEUR ############################################################
↪#####################################
########################################################################################
↪#####################################
# ici les nouvelles définitions/constantes utilisateur...
MA_CONSTANTE_UTILISATEUR = "Hello! =}"

########################################################################################
↪#####################################
# /CONFIGURATION UTILISATEUR ###########################################################
↪#####################################
########################################################################################
↪#####################################


#####################################
# DÉBUT CONFIGURATION CORELIBS... ###
########################################################################################
↪#####################################
# CONFIGURATION LOCALE #################################################################
↪#####################################
########################################################################################
↪#####################################
# les formats français et anglais sont définis par défaut
DEFAULT_LOCALE = ["fr_FR", "en_US"]

# code page encoding pour les retour terminal des appels commandes DOS
DEFAULT_DOS_CMD_CP_ENCODING = "cp850"  # cf. https://en.wikipedia.org/wiki/Code_page_
↪850

# format des tailles avec leur valeur minimale pour l'affichage automatisé
DEFAULT_MIN_BYTE_SIZE_FORMAT = {
    "octet": {"min_size": 0},
    "Ko": {"min_size": 1},
    "Mo": {"min_size": 1},
    "Go": {"min_size": 1},
    "To": {"min_size": 0.5}
}
########################################################################################
↪#####################################
# /CONFIGURATION LOCALE ################################################################
↪#####################################
########################################################################################
↪#####################################


########################################################################################
↪#####################################
# CONFIGURATION PROJET #################################################################
↪#####################################
########################################################################################
↪#####################################
# nom structures modèles de répertoires par défaut
```

(continues on next page)

(continued from previous page)

```
DEFAULT_DIR_SCAFFOLDING = {
    "input": {  # dossier contenant toutes les données "entrées"
        "name": "__INPUTS__",
        "make": True
    },
    "output": {  # dossier contenant toutes les données "sorties"
        "name": "__OUTPUTS__",
        "make": True
    },
    "logs": {  # dossier contenant toutes les sorties "logs"
        "name": "__LOGS__",
        "make": True
    },
    "docs": {  # dossier contenant toutes les documentations/specs liées au projet
        "name": "__DOCS__",
        "make": True
    },
}
################################################################################
↪#################################
# /CONFIGURATION PROJET ########################################################
↪#################################
################################################################################
↪#################################


################################################################################
↪#################################
# CONFIGURATION LOG ############################################################
↪#################################
################################################################################
↪#################################
# forcer l'exécution des instructions suivantes même quand une erreur est levée␣
↪(Attention à son emploi...)
DEFAULT_CONTINUE_ON_ERROR = False

# affichage détaillé de la pile d'exécution
DEFAULT_STACK_TRACE = False

# redirection de l'affichage détaillé de la pile d'exécution vers la log
DEFAULT_STACK_TRACE_2_FILE = False

# affichage détaillé de la pile d'exécution
# par défaut, le style de couleur sera :
# • "lightbg" si environnement Jupyter, sinon "darkbg2" si DEFAULT_STACK_TRACE_2_FILE␣
↪est faux
# • "plaintext" si DEFAULT_STACK_TRACE_2_FILE est vrai
DEFAULT_STYLE_STACK_TRACE = "default"  # valeur possibles : plaintext, color, darkbg2␣
↪ou lightbg

# contexte du code source pour les affichage détaillés des logs
DEFAULT_CONTEXT_SOURCE_LINES = 3

# affichage informations en provenance du package corelibs
DEFAULT_VERBOSE = False

# nom fichier log par défaut dans le cas des entrées standards
```

(continues on next page)

(continued from previous page)

```
DEFAULT_STDIN_LOGS_NAME = "__STDIN_"

# extension par défaut des fichiers logs
DEFAULT_LOGS_EXTENSION = ".log"

# niveau d"affichage des logs, valeurs possibles :
# * log.DEBUG <=> 10
# * log.INFO <=> 20 (valeur par défaut)
# * log.WARNING <=> 30
# * log.ERROR <=> 40
# * log.CRITICAL <=> 50
#
# pour désactiver une alerte, on augmente son niveau, par exemple WARNING (30)
# et dans ce cas, les alertes de niveau DEBUG et INFO seront ignorées.
DEFAULT_LOG_LEVEL = log.INFO  # ou 20

# le style par défaut des LOG affichées dans la sortie standard
DEFAULT_FIELD_STYLES = {
    "asctime": {"color": 242, "bright": True},
    "hostname": {"color": "magenta"},
    "username": {"color": "yellow"},
    "levelname": {"color": 242, "bright": True},
    "levelno": {"color": 242, "bright": True},
    "lineno": {"color": "white"},
    "process": {"color": "white"},
    "name": {"color": "blue"},
    "module": {"color": "blue"},
    "programname": {"color": "cyan"},
    "thread": {"color": "white"},
    "filename": {"color": "blue"},
    "funcName": {"color": "blue"},
}

# les couleurs par défaut selon le niveau
DEFAULT_LEVEL_STYLES = {
    "critical": {
        "color": 255,
        "background": "red",
    },
    "error": {"color": "red"},
    "warning": {"color": "yellow"},
    "debug": {"color": "green"},
    "info": {"color": "cyan"},
    "notice": {"color": "magenta"},
    "spam": {"color": "green"},
    "success": {"color": "green"},
    "verbose": {"color": "blue"},
}

# le label par défaut de la log
# si DEFAULT_SHORT_LOG_LABEL alors affiche simplement le nom du programme qui a␣
↪généré la log
# sinon affichera le chemin complet avec le nom du programme source et cible qui ont␣
↪levé l"info ou l"alerte
DEFAULT_SHORT_LOG_LABEL = True

# le format par défaut de la log
```

(continues on next page)

(continued from previous page)

```
DEFAULT_LOG_FORMAT = \
    "%(asctime)s %(username)s@%(hostname)s - %(name)s" \
    + " [P%(process)d - T%(thread)d - %(filename)s:%(lineno)05d] • %(levelname)13s :
↪%(message)s"

# le timestamp par défaut de la log
DEFAULT_LOG_DATE_FORMAT = "%Y-%m-%d %H:%M:%S"
################################################################################
↪#################################
# /CONFIGURATION LOG ###########################################################
↪#################################
################################################################################
↪#################################
```

CHAPTER

TWO

MODULE CLEANSE

Description générale

Module pour purifier/nettoyer les données

corelibs.cleanse.**is_datetime**(*dt_str*, *in_format='%d/%m/%Y %H:%M:%S'*, *out_format='%d/%m/%Y %H:%M:%S'*, *continue_on_error=False*)

Description

Vérifie si une chaîne est un datetime ou non.
Permet également de convertir le datetime vers un format souhaité (cf. pour les formats)

Parameters

- **dt_str** – datetime (chaine de caractère ou instance datetime)
- **in_format** – indique le format d'entrée (n'est pas nécessaire si le dt_str est une instance de datetime)
- **out_format** – indique le format sortie
- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

chaine vide (si forcé)
ou
None (si forcé lorsque dt_str est une instance de datetime)
ou

datetime au format out_format

Exemple :

```
# is_datetime.py
from corelibs import cleanse as cls
import datetime as dt


# forcer le traitement des instructions suivantes même en erreur...
print(cls.is_datetime("Hello Kim", continue_on_error=True))

# Datetime en chaine de caractère
# forcer le retour à vide (utile pour les données loufoques de la morktu)
print(cls.is_datetime("2020-08-31", "%Y-%m-%d", out_format=""))  # retourne vide..
↪.

print(cls.is_datetime("2020-08-31", "%Y-%m-%d"))  # retourne 31/08/2020 00:00:00
print(cls.is_datetime("lun., 31 août 2020 15:57:43", "%a, %d %b %Y %H:%M:%S"))  #␣
↪31/08/2020 15:57:43

# format sortie différente
print(cls.is_datetime("lun., 31 août 2020 15:57:43",
                      in_format="%a, %d %b %Y %H:%M:%S",
                      out_format="le %A %d %b %Y à %H h et %M min"))  # le lundi␣
↪31 août 2020 à 15 h et 57 min


# Instance datetime
_now = dt.datetime.now()
# sortie par défaut quand c'est une date...
print(cls.is_datetime(_now))  # retourne 15/12/2020 20:57:08

# sortie avec conversion format
print(cls.is_datetime(_now, out_format="le %A %d %b %Y à %H h et %M min"))  # le␣
↪mardi 15 déc. 2020 à 20 h et 58 min
```

corelibs.cleanse.**is_str**(*string*, *strip_non_printable_char=True*, *strip_accented_char=False*, *strip_num_char=False*, *chars_2_replace=None*, *replaced_chars=None*, *unicode_categories_2_remove=None*, *check_only=False*)

Description

Vérifie si une chaîne de caractères est bien une chaine de caractères.
Permet également de faire du nettoyage de caractères

Parameters

- **string** – chaine de caractères à traiter
- **strip_non_printable_char** – permet de nettoyer la chaine de tous les caractères de contrôles parasites.

valeurs possibles: *False/True*
valeur par défaut: *True*

- **strip_accented_char** – permet de nettoyer la chaine de tous les caractères accentués, avec leurs équivaleuts sans accent.

  valeurs possibles: *False/True*
  valeur par défaut: *False*

- **strip_num_char** – permet de nettoyer la chaine de tous les caractères numériques parasites (avec ou non des caractères séparateurs, ainsi que les signes et/ou opérateur comme le "-" ou ":")

  valeurs possibles: *False/True*
  valeur par défaut: *False*

- **chars_2_replace** – permet de faire du remplacement de caractères unitairement.

  valeurs possibles: *Chaine de caractère ou Dictionnaire associatif* (si Dictionnaire, alors l'argument replaced_chars est optionnel, cf. exemple)

- **replaced_chars** – indique la liste des caractères de remplacement, si cet argument est renseigné, doit fonctionner conjointement avec l'argument chars_2_replace (cf. exemple)
- **unicode_categories_2_remove** – permet de spécifier une classe de caractères Unicode à supprimer. Cet argument est optionnel et doit être utilisé avec l'argument strip_non_printable_char

  valeurs possibles: *Liste ou Tuple de classe Unicode* (cf. pour plus de détails sur les classifications des caractères Unicodes)
  valeur par défaut: *None*

- **check_only** – permet de retourner des booléens au lieu de la chaîne de caractères transformée (utile dans le cas où il est seulement souhaité un contrôle, sans nécessairement une transformation pour éventuellement y appliquer une règle de gestion propre)

Returns

chaine de caractères transformée si check_only == False (comportement par défaut)
sinon True/False

Exemple :

```
# is_str.py
from corelibs import cleanse as cls


# test si chaine caractères
print(cls.is_str(123))  # affichera une chaîne vide ""

# suppression de tous les caractères non imprimables
# \x00 <=> NULL
# \x01 <=> START OF HEADING (SOH)
# \x02 <=> START OF TEXT (STX)
# \x03 <=> END OF TEXT (ETX)
# \x04 <=> END OF TRANSMISSION (EOT)
# \x05 <=> END OF QUERY (ENQ)
# \x06 <=> ACKNOWLEDGE (ACK)
# \x07 <=> BEEP (BEL)
# \x08 <=> BACKSPACE (BS)
# \x09 <=> HORIZONTAL TAB (HT)
# \x0A <=> LINE FEED (LF)
# \x0B <=> VERTICAL TAB (VT)
# \x0C <=> FF (FORM FEED)
# \x0D <=> CR (CARRIAGE RETURN)
# \x0E <=> SO (SHIFT OUT)
# \x0F <=> SI (SHIFT IN)
# \x10 <=> DATA LINK ESCAPE (DLE)
# \x11 <=> DEVICE CONTROL 1 (DC1)
# \x12 <=> DEVICE CONTROL 2 (DC2)
# \x13 <=> DEVICE CONTROL 3 (DC3)
# \x14 <=> DEVICE CONTROL 4 (DC4)
# \x15 <=> NEGATIVE ACKNOWLEDGEMENT (NAK)
# \x16 <=> SYNCHRONIZE (SYN)
# \x17 <=> END OF TRANSMISSION BLOCK (ETB)
# \x18 <=> CANCEL (CAN)
# \x19 <=> END OF MEDIUM (EM)
# \x1A <=> SUBSTITUTE (SUB)
# \x1B <=> ESCAPE (ESC)
# \x1C <=> FILE SEPARATOR (FS) RIGHT ARROW
# \x1D <=> GROUP SEPARATOR (GS) LEFT ARROW
# \x1E <=> RECORD SEPARATOR (RS) UP ARROW
# \x1F <=> UNIT SEPARATOR (US) DOWN ARROW
chaine = "\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F\x10\
↪\x11\x12\x13\x14\x15\x16\x17\x18\x19" \
         "\x1A\x1B\x1C\x1D\x1E\x1F Hello, je suis la chaîne qui ne sera pas␣
↪nettoyée... \x12\x13\x14\x15\x16\x17\x18"
print(cls.is_str(chaine))  # affiche " Hello, je suis la chaîne qui ne sera pas␣
↪nettoyée... "

# Supression via des classes de caractères Unicode
# Cc => Caractères de contrôles (non imprimables)
# Lu => Caractères majuscules
# Zs => espace
print(cls.is_str(chaine, unicode_categories_2_remove=("Cc", "Lu", "Zs"))) #␣
↪affiche "ello,jesuislachaînequineserapasnettoyée..."

# suppression des caractères accentués en les ramplaçant par leurs équivalents␣
↪non accentués
chaine = "Hello, voici ma liste de caractères accentués␣
↪ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõöøùúûüýþÿŒœŠšŸƒ"
```

(continues on next page)

(continued from previous page)

```
print(cls.is_str(chaine, strip_accented_char=True))  # affiche "Hello, voici ma␣
↪liste de caracteres accentues␣
↪AAAAAAÆCEEEEIIIIÐNOOOOOØUUUUYÞßaaaaaaæceeeeiiiiðnoooooøuuuuyþyŒœSsYƒ"

# suppression des caractères numériques avec les signes, opérateurs et séparateurs
chaine = "Hello, voici ma liste avec des caractères numériques : date(28/11/2013)␣
↪heure(19:31) normal(123456) " \
         "séparateur (012.2345.12) signe + 12.2345.12 separateur 12,123,456 - 123␣
↪separateur 10 123 456 " \
         "signe - 123.12 signe et sep -12,123 signe et sep -10 012,123 1456"
print(cls.is_str(chaine, strip_num_char=True))  # affiche "Hello, voici ma liste␣
↪avec des caractères numériques : date() heure() normal() séparateur () signe␣
↪separateur separateur signe signe et sep signe et sep"

# remplacement des caractères, K -> A, i -> n et m -> T 3 -> 7
# 1er exemple avec une liste de caractères
chaine = "Hello, ma \x12\x13\x14\x15\x16\x17\x18 chaine avec des caractères à␣
↪remplacer : Kim Marie Adélie, ma petite choupinette a 3 ans (casse prise en␣
↪coMpte...)"
print(cls.is_str(chaine, chars_2_replace="Kim3", replaced_chars="AnT7"))  #␣
↪affiche "Hello, Ta  channe avec des caractères à reTplacer : AnT Marne Adélne,␣
↪Ta petnte choupnnette a 7 ans (casse prnse en coMpte...)"

# 2ème exemple avec un dictionnaire
print(cls.is_str(chaine,
                 chars_2_replace={
                     "K": "A",
                     "i": "n",
                     "m": "T",
                     "3": "7"
                 }))  # Hello, Ta  channe avec des caractères à reTplacer : AnT␣
↪Marne Adélne, Ta petnte choupnnette a 7 ans (casse prnse en coMpte...)
```

CHAPTER

THREE

MODULE LAZY

Description générale

Module de base avec des fonctions, décorateurs utiles, etc…

corelibs.lazy.**add_dir_path_2_project**(*path*)

Description

Permet d'inclure dans le projet actuel des programmes python tiers, enregistrés dans un emplacement différent.

Warning:

En principe, il est très rare de faire appel à cette fonction. Si cela se répète, il faudrait éventuellement revoir la structure du projet.

Parameters **path** – indique l'emplacement en chemin absolu du **dossier** contenant le ou les programmes python à inclure dans le projet actuel.

Returns

rien…

Exemple :

```
# D:\OneDrive\Documents\_TEST_\PY_2_IMPORT\programme_importe.py


# Les instructions ci-dessous n'appartiennent pas au projet... et son emplacement␣
↪source est D:\OneDrive\Documents\_TEST_\PY_2_IMPORT
def say_hello(who):
        print("Hello", who)
```

```
# add_dir_path_2_project.py
from corelibs import lazy as lz

# inclusion du dossier parent contenant les programmes à inclure, ici D:\OneDrive\
↪Documents\_TEST_\PY_2_IMPORT
lz.add_dir_path_2_project(r"D:\OneDrive\Documents\_TEST_\PY_2_IMPORT")

try:
    import programme_importe as pi
except ImportError:
    raise Exception("\n\nProblème import programme tiers d'un emplacement␣
↪loufoque, hors projet")

pi.say_hello("Kim!")  # affichera Hello Kim! si tout est OK
```

corelibs.lazy.**convert_2_dict**(*obj*)

Description

Permet de convertir tous les objets (convertissables) en dictionnaire de manière récursive.

Parameters **obj** – objet à convertir

Returns

dictionnaire
ou
objet contenant un dictionnaire imbriqué converti

Exemple :

```
# convert_2_dict.py
from collections import namedtuple
from corelibs import lazy as lz


# création d'un objet PuPuce tuple nommé
PuPuce = namedtuple("PuPuce", ["name", "age"])
kim = PuPuce(
    name="Kim",
    age=6
)
print(kim)  # affiche PuPuce(name='Kim', age=6)

print(lz.is_namedtuple_instance(kim))  # affiche bien True

# conversion en dictionnaire
print(lz.convert_2_dict(kim))  # affiche {'name': 'Kim', 'age': 6}
```

(continues on next page)

(continued from previous page)

```
# création d'un tuple nommé imbriqué
NestedPuPuce = namedtuple("NestedPuPuce", ["name", "age", "nested_tuple"])
kim2 = NestedPuPuce(
    name="Kim",
    age=6,
    nested_tuple=kim
)
print(kim2)  # affiche NestedPuPuce(name='Kim', age=6, nested_tuple=PuPuce(name=
↪'Kim', age=6))

# conversion en dictionnaire
print(lz.convert_2_dict(kim2))  # {'name': 'Kim', 'age': 6, 'nested_tuple': {'name
↪': 'Kim', 'age': 6}}

# tableau non convertissable
print(lz.convert_2_dict(["TRUONG", "Kim", 6]))  # ['TRUONG', 'Kim', 6]

# conversion d'un tuple nommé au sein du tableau...
print(lz.convert_2_dict(["TRUONG", "Kim", 6, kim2]))  # ['TRUONG', 'Kim', 6, {
↪'name': 'Kim', 'age': 6, 'nested_tuple': {'name': 'Kim', 'age': 6}}]
```

corelibs.lazy.**copy**(*source*, *destination*)

Description

Permet de copier un ou des fichiers vers une nouvelle destination ou des nouvelles destinations. Les fichiers sont au sens Unix du terme (i.e. soit fichier régulier, soit répertoire)

Warning:

La copie lèvera une alerte si des sous répertoires destinations existent et portent les mêmes noms que les sous répertoire sources.

Parameters

- **source** – indique l'emplacement source du ou des fichiers à copier avec le(s) chemin(s) absolu(s), avec ou sans schéma.
- **destination** – indique l'emplacement destination du ou des fichiers à copier avec le(s) chemin(s) absolu(s).

Returns

rien...

Exemple :

```
# copy.py
from corelibs import lazy as lz


# Création dossier destination
repertoire_destination = r"D:\OneDrive\Documents\_TEST_\_COPY_DESTINATION_"
lz.mkdir(repertoire_destination, make_scaffolding=False)

# Copie simple de fichier standard, sans renommage
lz.copy(r"\\wsl$\Ubuntu-20.04\root\.zsh_history", repertoire_destination)  #␣
↪chemin réseau...

# Copie simple de fichier standard, avec renommage
lz.copy(r"D:\OneDrive\Documents\_TEST_\_éèçàoöôîïêëùûü;.txt", repertoire_
↪destination + "\\nouveau_nom.txt")

# Copie simple de répertoire standard, sans renommage
lz.copy(r"D:\OneDrive\Documents\_TEST_\__R2D2-LOGS__", repertoire_destination + "\
↪\__R2D2-LOGS__")  # IMPORTANT!!! remettre le même nom de dossier destination,␣
↪autrement, la copie se fera au niveau du répertoire parent

# Copie simple de répertoire standard, avec renommage
lz.copy(r"D:\OneDrive\Documents\_TEST_\__R2D2-LOGS__", repertoire_destination + "\
↪\__R2D2__")

# Copie via mode modèle
lz.copy(r"D:\OneDrive\Documents\_TEST_\*.sas*", repertoire_destination)  # modèle␣
↪avec extension
# ou
lz.copy(r"D:\OneDrive\Documents\_TEST_\*2020-11-11*", repertoire_destination)  #␣
↪modèle sans extension, comprenant la chaîne "2020-11-11" dans le nom

# Copie groupée dans un dossier
lz.copy((
    r"D:\OneDrive\Documents\_TEST_\*.sas*",  # avec schéma
    r"D:\OneDrive\Documents\_TEST_\2020-11-11.jpg",
    r"D:\OneDrive\Documents\_TEST_\__R2D2-LOGS__"  # dossier...
), repertoire_destination)

# Copie groupée d'une liste de fichiers dans une autre liste de fichiers␣
↪(fonctionnement 1-1, i.e. même nombre de fichiers en entrée et en sortie,␣
↪traitée de manière itérative)
lz.copy((
    r"D:\OneDrive\Documents\_TEST_\*.sas*",  # avec schéma
    r"D:\OneDrive\Documents\_TEST_\2020-11-11.jpg",
    r"D:\OneDrive\Documents\_TEST_\__R2D2-LOGS__",  # dossier...
), (
    repertoire_destination,  # sans renommage
    repertoire_destination + "\\2020-11-11_NOUVEAU_NOM.jpg",  # avec renommage
    repertoire_destination + "\\__R2D2-NEW__",  # avec renommage
))
```

corelibs.lazy.**datetime_2_epoch**(*date_time*, *time_format='%d/%m/%Y %H:%M:%S'*, *reference_epoch='Unix'*, *continue_on_error=False*)

Description

Permet de convertir une date/heure en nombre de secondes écoulés depuis le temps de référence epoch.

cf. *corelibs.lazy.epoch_2_datetime()* pour la conversion inverse

---

Parameters

- **date_time** – date à convertir
- **time_format** – indique le format de la date et heure

  valeur par défaut: "%d/%m/%Y %H:%M:%S" (DD/MM/AAAA HH:MM:SS)

- **reference_epoch** – indique l'époque de référence

  valeurs possibles: "Unix", "Windows"
  valeur par défaut: "Unix"

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

timestamp au format epoch

Exemple :

```
# datetime_2_epoch.py
from corelibs import lazy as lz

# conversion en epoch Unix (https://www.epochconverter.com/#tools)
print(lz.datetime_2_epoch(date_time=" Oct 3 1978       1:33PM  ", time_format="%b
↪%d %Y %I:%M%p"))  # même si ce type de chaîne loufoque est nettoyée avant le␣
↪calcul, il est préférable d'écrire proprement la donnée...
# 276269580000
print(lz.datetime_2_epoch("28/11/2013 19:23:52"))
# 1385666632000

# conversion en epoch Windows (https://www.epochconverter.com/ldap)
print(lz.datetime_2_epoch(date_time="Oct 3 1978 1:33PM", time_format="%b %d %Y %I:
↪%M%p", reference_epoch="Windows"))
# 119207431800000000
```

(continues on next page)

(continued from previous page)

```
print(lz.datetime_2_epoch(date_time="28/11/2013 19:23:52", reference_epoch=
↪"Windows"))
# 130301402320000000
```

corelibs.lazy.**delete_files** (*file_path*, *extension=''*, *remove_empty_dir=True*, *verbose=False*)

Description

Permet de supprimer des fichiers ou répertoires récursivement. La suppression peut se faire également en se basant sur plusieurs extensions différentes (cf. exemple)

Parameters

- **file_path** – le chemin absolu du fichier ou du répertoire à supprimer
- **extension** – le(s) extension(s) ou modèle des fichiers à supprimer

  valeurs possibles: *expression régulière* pour désigner des extensions/modèles de nom de fichiers
  valeur par défaut: *rien* (pour éviter les erreurs. . .)

- **remove_empty_dir** – indique si il faut supprimer ou non le répertoire vidé

  valeurs possibles: *False/True*
  valeur par défaut: *True*

- **verbose** – afficher ou non les messages d'info/alertes

  valeurs possibles: *False/True*
  valeur par défaut: *DEFAULT_VERBOSE* (cf. *Configurations*)

Returns

rien. . .

Exemple :

```
# delete_files.py
from corelibs import lazy as lz
```

(continues on next page)

(continued from previous page)

```
# Suppression d'un fichier ciblé et nommé "Nouveau document texte.txt"
file_2_del = r"D:\OneDrive\Documents\_TEST_\DosASupp\Nouveau document texte.txt"
lz.delete_files(file_2_del, verbose=True)


# Suppression de tous les fichiers avec une extension .RTF
files_2_del = r"D:\OneDrive\Documents\_TEST_\DosASupp"
lz.delete_files(files_2_del, extension="*.rtf", verbose=True)


# Suppression de tous les fichiers avec une extension .DOC* et .XLS*
files_2_del = r"D:\OneDrive\Documents\_TEST_\DosASupp"
lz.delete_files(files_2_del, extension="*.doc*,*.xls*", verbose=True)


# L'extension peut prendre un modèle regex, par exemple :
# Suppression de tous les fichiers ayant le mot _LOG_ dans le nom
files_2_del = r"D:\OneDrive\Documents\_TEST_\DosASupp"
lz.delete_files(files_2_del, extension="*_LOG_*", verbose=True)


# Suppression complète
#   • tous les fichiers à l'intérieur du dossier DosASupp
#   • une fois vide, le dossier DosASupp est supprimé
folder_2_del = r"D:\OneDrive\Documents\_TEST_\DosASupp"
lz.delete_files(folder_2_del, extension="*", verbose=True)
```

corelibs.lazy.**epoch_2_datetime**(*seconds*, *time_format='%d/%m/%Y %H:%M:%S'*)

Description

Permet de convertir un nombre de secondes vers un format date/heure spécifié. L'epoch est le temps initial de référence, à partir duquel on mesure les secondes écoulées pour calculer les dates/heures. Sous UNIX/POSIX, ce temps correspond au 1 janvier 1970 00:00:00 UT et sous Windows NT, 1 janvier 1601 00:00:00 UT.

cf. *corelibs.lazy.datetime_2_epoch()* pour la conversion inverse

---

Parameters

- **seconds** – indique le nombre de secondes
- **time_format** – indique le format de sortie souhaité

  valeur par défaut: "%d/%m/%Y %H:%M:%S" (DD/MM/AAAA HH:MM:SS)

Returns

timestamp
ou
None (si problème)

Exemple :

```
# epoch_2_datetime.py
from corelibs import lazy as lz

print(lz.epoch_2_datetime(1605050212))
# 11/11/2020 00:16:52
```

corelibs.lazy.**get_abspath**(*root_dir_path*, *dir_2_join*)

Description

Retourne le chemin absolu normalisé à partir d'un couple (chemin, dossier)

Parameters

- **root_dir_path** – le chemin absolu
- **dir_2_join** – le dosssier à concaténer au chemin absolu

Returns

le chemin normalisé

Exemple :

```
# get_abspath.py
from corelibs import lazy as lz

abs_path = lz.get_abspath(r"C:\documents/dir", "toto")
print("Le chemin absolu normalisé est \"{abs_path}\"".format(abs_path=abs_path))
```

Terminal :

```
$ python get_abspath.py
Le chemin absolu normalisé est "C:\documents\dir\toto"
```

corelibs.lazy.**get_bytes_size_4_human**(*byte_size_format*, *default_format=None*, *min_byte_size_format={'Go': {'min_size': 1}, 'Ko': {'min_size': 1}, 'Mo': {'min_size': 1}, 'To': {'min_size': 0.5}, 'octet': {'min_size': 0}}*, *size_unit=True*, *continue_on_error=False*)

Description

Permet de lister un tuple nommé contenant toutes les valeurs converties à partir d'une taille en octet.

Parameters

- **byte_size_format** – tuple nommé calculé par *corelibs.lazy.get_bytes_size_formats()*
- **default_format** – format d'affichage souhaité | valeur par défaut: None, laissant le choix à la fonction de retourner la meilleure valeur | valeurs possible: "octet", "Ko", "Mo", "Go" ou "To"
- **min_byte_size_format** – lorsque le format d'affichage *default_format* est à None alors sera calculé automatiquement le meilleur format à afficher, dont les seuils minimums sont définis dans *DEFAULT_MIN_BYTE_SIZE_FORMAT* (cf. *Configurations*)
- **size_unit** – afficher l'unité de mesure

  valeurs possibles: *False/True*
  valeur par défaut: *True*

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

string sous la forme XX.XX Unité (où Unité est octet, Ko, Mo, Go ou To)
ou
float (si l'unité de mesure n'est pas souhaitée)

Exemple :

```
# get_bytes_size_formats.py
from corelibs import lazy as lz


print(lz.get_bytes_size_formats(0))
# affiche ByteSize(byte=0, kilobyte=0.0, megabyte=0.0, gigabyte=0.0, terabyte=0.0)
print(lz.get_bytes_size_formats(153800565))
# affiche ByteSize(byte=153800565, kilobyte=150195.86, megabyte=146.68,␣
↪gigabyte=0.14, terabyte=0.0)
print(lz.get_bytes_size_formats(1739886085))
# affiche ByteSize(byte=1739886085, kilobyte=1699107.5, megabyte=1659.28,␣
↪gigabyte=1.62, terabyte=0.0)
```

corelibs.lazy.**get_bytes_size_formats**(*byte_size*)

Description

Permet de lister un tuple nommé contenant toutes les valeurs converties à partir d'une taille en octet.

Parameters **byte_size** – indique le nombre d'octets

Returns

tuple nommé avec comme attributs :

- byte
- kilobyte
- megabyte
- gigabyte
- terabyte

Exemple :

```
# get_bytes_size_formats.py
from corelibs import lazy as lz


print(lz.get_bytes_size_formats(0))
# affiche ByteSize(byte=0, kilobyte=0.0, megabyte=0.0, gigabyte=0.0, terabyte=0.0)
print(lz.get_bytes_size_formats(153800565))
# affiche ByteSize(byte=153800565, kilobyte=150195.86, megabyte=146.68,␣
↪gigabyte=0.14, terabyte=0.0)
print(lz.get_bytes_size_formats(1739886085))
# affiche ByteSize(byte=1739886085, kilobyte=1699107.5, megabyte=1659.28,␣
↪gigabyte=1.62, terabyte=0.0)
```

corelibs.lazy.**get_caller_line_number**(*back_level=3*, *continue_on_error=False*)

Description

Permet de connaître la ligne du module source appelant un autre module.

Parameters

- **back_level** – profondeur d'appel de la fonction parent/enfant

valeurs possibles: *entier*
valeur par défaut: *3*

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns ligne du module appelant

Exemple :

```
# get_caller_line_number.py
from corelibs import config, lazy as lz


# appel croisé correct...
def info(profondeur_appel=1):
    return lz.get_caller_line_number(profondeur_appel)


print(
    "Le numéro de la ligne du module appelant est {lineno}"
    .format(
        lineno=info(profondeur_appel=1)  # affichera "Le numéro de la ligne du␣
↪module appelant est 7"
    )
)

print(
    "Le numéro de la ligne du module appelant est {lineno}"
    .format(
        lineno=info(profondeur_appel=2)  # affichera "Le numéro de la ligne du␣
↪module appelant est 20"
    )
)

# appel direct incorrect...
lz.get_caller_line_number()
```

corelibs.lazy.**get_caller_module_name**()

Description

Permet de connaître le nom du module source appelant un autre module.

Returns nom du module appelant

Exemple :

```
# get_caller_module_name.py
from corelibs import lazy as lz


def get_module_info():
    caller_name = lz.get_caller_module_name()
    print("Le nom du module python appelant est \"{caller_name}\"".format(caller_
↪name=caller_name))
```

```
# caller_module_name.py
from tests.lazy import get_caller_module_name as gcmn


gcmn.get_module_info()
```

Terminal :

```
$ python caller_module_name.py
Le nom du module python appelant est "caller_module_name.py"
```

corelibs.lazy.**get_closest_value_in_list**(*value*, *list_of_values*, *continue_on_error=False*)

Description

Permet de récupérer la valeur la plus proche (en delta absolu) dans une liste de valeurs pour une valeur donnée.

Parameters

- **value** – indique la valeur de référence
- **list_of_values** – liste de valeurs (tuples ou tableaux)
- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

valeur la plus proche trouvée

Exemple :

```
# get_closest_value_in_list.py
from corelibs import lazy as lz

print(
    lz.get_closest_value_in_list(1.6, [1, 2, 3])
)  # retourne 2 qui est la valeur la plus proche de 1.6 en delta absolu

print(
    lz.get_closest_value_in_list(74.7, (10, 20, 30, 40, 50, 60, 70, 80, 90, 100))
)  # retourne 70


array = [2, 42, 82, 122, 162, 202, 242, 282, 322, 362]
number = 103
print(lz.get_closest_value_in_list(number, array))  # retourne 122
```

corelibs.lazy.**get_dir_n_basename**(*path*)

Description

Retourne un tuple (chemin, nom fichier ou nom répertoire) à partir d'un chemin absolu normalisé (e.g. "C:\documents\dir" retournera "dir" et "C:\documents\dir\fichier.txt" retournera "fichier.txt")

Parameters **path** – chemin absolu normalisé

Returns

tuple nommé avec comme attributs :

- dir_path
- base_name

Exemple :

```
# get_dir_n_basename.py
from corelibs import lazy as lz


# chemin + fichier (si chemin seul, sera retourné alors le nom du dossier + le␣
↪chemin amenant au dossier)
dir_n_basename = lz.get_dir_n_basename(r"C:\Users\M47624\corelibs\tests\lazy\get_
↪dir_n_basename.py")
# récupération par index
print("Le chemin est \"{dir}\"".format(dir=dir_n_basename[0]))
print("Le fichier est \"{base}\"".format(base=dir_n_basename[1]))
# récupération par attributs
print("Le chemin est \"{dir}\"".format(dir=dir_n_basename.dir_path))
print("Le fichier est \"{base}\"".format(base=dir_n_basename.base_name))

# sortie "séparée" ou "déballée" (unpacked)
dir, base = lz.get_dir_n_basename(r"C:\Users\M47624\corelibs\tests\lazy\get_dir_n_
↪basename.py")
```

(continues on next page)

(continued from previous page)

```
print("Le chemin est \"{dir}\"".format(dir=dir))
print("Le fichier est \"{base}\"".format(base=base))
```

Terminal :

```
$ python get_dir_n_basename.py
Le chemin est "C:\Users\M47624\corelibs\tests\lazy"
Le fichier est "get_dir_n_basename.py"
Le chemin est "C:\Users\M47624\corelibs\tests\lazy"
Le fichier est "get_dir_n_basename.py"
Le chemin est "C:\Users\M47624\corelibs\tests\lazy"
Le fichier est "get_dir_n_basename.py"
```

corelibs.lazy.**get_file_extension**(*filename*, *extensions=True*, *split_extensions=False*)

Description

Retourne un tuple (nom fichier, .extension(s))

Parameters

- **filename** – nom du fichier avec extension (avec ou sans chemin)
- **extensions** – précise si le fichier comporte des extensions composites ou non (e.g. ".tar.gz")

  valeurs possibles: *False/True*
  valeur par défaut: *True*

- **split_extensions** – dans le cas d'une extension composite (e.g. ".tar.gz") renvoie soit une chaine de caractère (i.e. ".tar.gz"), soit un tableau d'extensions (i.e. [".tar", ".gz"])

  valeurs possibles: *False/True*
  valeur par défaut: *False* (renvoie par défaut une chaine d'extensions composites)

Returns

tuple nommé avec comme attributs :

- file_name
- file_extension

Exemple :

```
# get_file_extension.py
from corelibs import lazy as lz

stem, suffix = lz.get_file_extension(r"\\file\path\file name.tar.gz")
print("Le nom du fichier sans extension est \"{stem}\"".format(stem=stem))
# Le nom du fichier sans extension est "file name"
print("Le nom de l'extension par défaut est \"{suffix}\"".format(suffix=suffix))
# Le nom de l'extension par défaut est ".tar.gz"

stem, suffix = lz.get_file_extension(r"\\file\path\file name.tar.gz", split_
↪extensions=True)
print("Le nom du fichier sans extension est \"{stem}\"".format(stem=stem))
# Le nom du fichier sans extension est "file name"
print("Le tableau des extensions est \"{suffix}\"".format(suffix=suffix))
# Le tableau des extensions est "['.tar', '.gz']"

file_properties = lz.get_file_extension(r"\\file\path\file name.tar.gz", split_
↪extensions=True)
print("Le nom du fichier sans extension est \"{stem}\"".format(stem=file_
↪properties.file_name))
# Le nom du fichier sans extension est "file name"
print("Le tableau des extensions est \"{suffix}\"".format(suffix=file_properties.
↪file_extension))
# Le tableau des extensions est "['.tar', '.gz']"
```

corelibs.lazy.**get_home**()

Description

Permet de retrouver le chemin de l'utilisateur ("home")

Returns

le chemin absolu du home

Exemple :

```
# get_home.py
from corelibs import lazy as lz

print(lz.get_home())
```

corelibs.lazy.**get_hostname**()

Description

Récupère le nom de l'ordinateur courant

Returns

nom de l'ordinateur courant

Exemple :

```
# get_hostname.py
from corelibs import config, lazy as lz

hostname = lz.get_hostname()
print("Le nom de l'ordinateur actuel est \"{hostname}\"".
↪format(hostname=hostname))
```

corelibs.lazy.**get_module_name**()

Description

Retourne le nom du module courant

Returns nom module courant

Exemple :

```
# get_module_name.py
from corelibs import lazy as lz

module_name = lz.get_module_name()
print("Le nom du module python est \"{module_name}\"".format(module_name=module_
↪name))
```

corelibs.lazy.**get_module_path**()

Description

Retourne le chemin du script/programme python courant

Returns

os.path.realpath(sys.argv[0])
ou
os.path.dirname(os.path.realpath(sys.argv[0])))

Exemple :

```
# get_module_path.py
from corelibs import lazy as lz

path = lz.get_module_path()
print("Le chemin du programme python est \"{path}\"".format(path=path))
```

Terminal :

```
$ python get_module_path.py
Le chemin du programme python est "C:\Users\M47624\corelibs\tests\lazy"
```

corelibs.lazy.**get_path_docs_dir**(*location=None*)

Description

Retourne le chemin absolu du répertoire des docs/specs, nommé par défaut "__DOCS__" cf. *corelibs.lazy.mkdir()* pour plus de détails.

Parameters **location** – la location du répertoire racine contenant le répertoire des docs/specs.

valeur par défaut: le chemin retourné par *corelibs.lazy.get_module_path()*

Returns

chemin absolu du répertoire "__DOCS__" (si existe, e.g. "C:\Users\M47624__DOCS__")
ou
chaîne vide (sinon)

Exemple :

```
# get_path_docs_dir.py
from corelibs import config, lazy as lz

root_path = r"D:\OneDrive\Documents\_TEST_\PARENTS\INEXISTANTS"

print(lz.get_path_docs_dir(root_path))  # retourne rien

lz.mkdir(
```

(continues on next page)

(continued from previous page)

```
    root_path,
    dir_scaffolding={
        "input": {  # dossier contenant toutes les données "entrées"
            "name": "__R2 D2__",
            "make": False
        },
        "output": {  # dossier contenant toutes les données "sorties"
            "name": "__MY_OUTPUTS__",
            "make": False
        },
        "logs": {  # dossier contenant toutes les sorties "logs"
            "name": "__MY-LOGS__",
            "make": False
        },
        "docs": {  # dossier contenant toutes les documentations/specs liées au␣
↪projet
            "name": "__SPECS__",
            "make": True
        },
    },
    verbose=False
)
print(lz.get_path_docs_dir(root_path))  # retourne "D:\OneDrive\Documents\_TEST_\
↪PARENTS\INEXISTANTS\__SPECS__"
```

corelibs.lazy.**get_path_input_dir**(*location=None*)

Description

Retourne le chemin absolu du répertoire des entrées, nommé par défaut "__INPUTS__" cf. *corelibs.lazy.mkdir()* pour plus de détails.

Parameters **location** – la location du répertoire racine contenant le répertoire des entrées.

valeur par défaut: le chemin retourné par *corelibs.lazy.get_module_path()*

Returns

chemin absolu du répertoire "__INPUTS__" (si existe, e.g. "C:\Users\M47624__INPUTS__")
ou
chaîne vide (sinon)

Exemple :

```
# get_path_input_dir.py
from corelibs import config, lazy as lz

root_path = r"D:\OneDrive\Documents\_TEST_\PARENTS\INEXISTANTS"

print(lz.get_path_input_dir(root_path))  # retourne rien

lz.mkdir(
    root_path,
    dir_scaffolding={
        "input": {  # dossier contenant toutes les données "entrées"
            "name": "__R2 D2__",
            "make": True
        },
        "output": {  # dossier contenant toutes les données "sorties"
            "name": "__MY_OUTPUTS__",
            "make": False
        },
        "logs": {  # dossier contenant toutes les sorties "logs"
            "name": "__MY-LOGS__",
            "make": False
        },
        "docs": {  # dossier contenant toutes les documentations/specs liées au␣
↪projet
            "name": "__DOCS__",
            "make": False
        },
    },
    verbose=False
)
print(lz.get_path_input_dir(root_path))  # retourne "D:\OneDrive\Documents\_TEST_\
↪PARENTS\INEXISTANTS\__R2 D2__"
```

corelibs.lazy.**get_path_logs_dir**(*location=None*)

Description

Retourne le chemin absolu du répertoire des logs, nommé par défaut "__LOGS__" cf. *corelibs.lazy.mkdir()* pour plus de détails.

Parameters **location** – la location du répertoire racine contenant le répertoire des logs.

valeur par défaut: le chemin retourné par *corelibs.lazy.get_module_path()*

Returns

chemin absolu du répertoire "__LOGS__" (si existe, e.g. "C:\Users\M47624__LOGS__")
ou

chaîne vide (sinon)

Exemple :

```
# get_path_logs_dir.py
from corelibs import config, lazy as lz

root_path = r"D:\OneDrive\Documents\_TEST_\PARENTS\INEXISTANTS"

print(lz.get_path_logs_dir(root_path))  # retourne rien

lz.mkdir(
    root_path,
    dir_scaffolding={
        "input": {  # dossier contenant toutes les données "entrées"
            "name": "__R2 D2__",
            "make": False
        },
        "output": {  # dossier contenant toutes les données "sorties"
            "name": "__MY_OUTPUTS__",
            "make": False
        },
        "logs": {  # dossier contenant toutes les sorties "logs"
            "name": "__MY-LOGS__",
            "make": True
        },
        "docs": {  # dossier contenant toutes les documentations/specs liées au␣
↪projet
            "name": "__DOCS__",
            "make": False
        },
    },
    verbose=False
)
print(lz.get_path_logs_dir(root_path))  # retourne "D:\OneDrive\Documents\_TEST_\
↪PARENTS\INEXISTANTS\__MY_LOGS__"
```

corelibs.lazy.**get_path_output_dir**(*location=None*)

Description

Retourne le chemin absolu du répertoire des sorties, nommé par défaut "__OUTPUTS__" cf. *corelibs.lazy.mkdir()* pour plus de détails.

Parameters **location** – la location du répertoire racine contenant le répertoire des sorties.

valeur par défaut: le chemin retourné par *corelibs.lazy.get_module_path()*

Returns

chemin absolu du répertoire “__OUTPUTS__” (si existe, e.g. “C:\Users\M47624__OUTPUTS__”)

ou

chaîne vide (sinon)

Exemple :

```
# get_path_output_dir.py
from corelibs import config, lazy as lz

root_path = r"D:\OneDrive\Documents\_TEST_\PARENTS\INEXISTANTS"

print(lz.get_path_output_dir(root_path))  # retourne rien

lz.mkdir(
    root_path,
    dir_scaffolding={
        "input": {  # dossier contenant toutes les données "entrées"
            "name": "__R2 D2__",
            "make": False
        },
        "output": {  # dossier contenant toutes les données "sorties"
            "name": "__MY_OUTPUTS__",
            "make": True
        },
        "logs": {  # dossier contenant toutes les sorties "logs"
            "name": "__MY-LOGS__",
            "make": False
        },
        "docs": {  # dossier contenant toutes les documentations/specs liées au␣
↪projet
            "name": "__DOCS__",
            "make": False
        },
    },
    verbose=False
)
print(lz.get_path_output_dir(root_path))  # retourne "D:\OneDrive\Documents\_TEST_
↪\PARENTS\INEXISTANTS\__MY_OUTPUTS__"
```

corelibs.lazy.**get_path_scaffold_directories**(*location=None*)

Description

Retourne les chemins absolus de tous les répertoires créés cf. *corelibs.lazy.mkdir()* pour plus de détails.

Parameters **location** – la location du répertoire racine contenant les répertoires modèles.

valeur par défaut: le chemin retourné par *corelibs.lazy.get_module_path()*

Returns

tuple nommé avec comme attributs :

- docs : chemin absolu du répertoire "__DOCS__" (si existe)
- input : chemin absolu du répertoire "__INPUTS__" (si existe)
- output : chemin absolu du répertoire "__OUTPUTS__" (si existe)
- logs : chemin absolu du répertoire "__LOGS__" (si existe)

Exemple :

```
# get_path_scaffold_directories.py
from corelibs import config, lazy as lz

root_path = r"D:\OneDrive\Documents\_TEST_\PARENTS\INEXISTANTS"

print(lz.get_path_scaffold_directories(root_path))  # retourne None

lz.mkdir(
    root_path,
    dir_scaffolding={
        "input": {  # dossier contenant toutes les données "entrées"
            "name": "__R2 D2__",
            "make": True
        },
        "output": {  # dossier contenant toutes les données "sorties"
            "name": "__MY_OUTPUTS__",
            "make": True
        },
        "logs": {  # dossier contenant toutes les sorties "logs"
            "name": "__MY-LOGS__",
            "make": True
        },
        "docs": {  # dossier contenant toutes les documentations/specs liées au␣
↪projet
            "name": "__DOCUMENTATIONS__",
            "make": True
        },
    },
    verbose=False
)

scaffold_dir_path = lz.get_path_scaffold_directories(root_path)
print(scaffold_dir_path)

# récupréer les chemins des docs
print(scaffold_dir_path.docs)

# récupréer les chemins des inputs
print(scaffold_dir_path.input)

# récupréer les chemins des outputs
print(scaffold_dir_path.output)

# récupréer les chemins des logs
print(scaffold_dir_path.logs)
```

(continues on next page)

(continued from previous page)

```
# si le définition est écrasée, alors le get_path_scaffold_directories() va␣
↪tenter de retrouver les nouvelles
# informations, par exemple ci-dessous, le nouveau nom du répertoire "input" s
↪'appellera "__R2 D2__" et ainsi de suite
config.DEFAULT_DIR_SCAFFOLDING = {
    "input": {  # dossier contenant toutes les données "entrées"
        "name": "__R2 D2__",
        "make": False
    },
    "output": {  # dossier contenant toutes les données "sorties"
        "name": "__MY_OUTPUTS__",
        "make": False
    },
    "logs": {  # dossier contenant toutes les sorties "logs"
        "name": "__MY-LOGS__",
        "make": True
    },
    "docs": {  # dossier contenant toutes les documentations/specs liées au projet
        "name": "__DOCS__",
        "make": False
    },
}

print(lz.get_path_scaffold_directories(root_path))  # affichera
# ScaffoldDir(input='D:\\OneDrive\\Documents\\_TEST_\\PARENTS\\INEXISTANTS\\__R2␣
↪D2__', output='D:\\OneDrive\\Documents\\_TEST_\\PARENTS\\INEXISTANTS\\__MY_
↪OUTPUTS__', logs='D:\\OneDrive\\Documents\\_TEST_\\PARENTS\\INEXISTANTS\\__MY-
↪LOGS__', docs='')
# "docs" est vide car nous cherchons un nom "__DOCS__" alors que nous avons créé
↪"__DOCUMENTATIONS__"
```

corelibs.lazy.**get_timestamp**(*timestamp_format='DT'*, *display_ms=False*, *only_ms=False*)

Description

Retourne un timestamp normalisé ; utile pour suffixer les noms des fichiers

---

Parameters

- **timestamp_format** – timestamp_format souhaité du timestamp | valeurs possibles: *DT, D, T, NOW, GD ou GT* | valeur par défaut: *DT*
- **display_ms** – afficher ou non les milli secondes

  valeurs possibles: *False/True*
  valeur par défaut: *False*

- **only_ms** – récupère seulement les millièmes de secondes (pratique si besoin d'un *seed*)

valeurs possibles: *False/True*
valeur par défaut: *False*

Returns

timestamp, selon timestamp_format :
* YYYYMMDD_HHMMSS.SSSSSS
* YYYYMMDD_HHMMSS
* ou SSSSSS

Exemple :

```
# get_timestamp.py
from corelibs import lazy as lz

timestamp = lz.get_timestamp()
print("Le timestamp sans milli secondes est {timestamp}".
↪format(timestamp=timestamp))
# Le timestamp sans milli secondes est 20201209_181434

timestamp = lz.get_timestamp(only_ms=True)
print("Le timestamp en milli secondes est {timestamp}".
↪format(timestamp=timestamp))
# Le timestamp en milli secondes est 096248

timestamp = lz.get_timestamp(display_ms=True)
print("Le timestamp avec milli secondes est {timestamp}".
↪format(timestamp=timestamp))
# Le timestamp avec milli secondes est 20201209_181434.097248

timestamp = lz.get_timestamp(timestamp_format="D")
print("Le timestamp avec la date seulement est {timestamp}".
↪format(timestamp=timestamp))
# Le timestamp avec la date seulement est 20201209

timestamp = lz.get_timestamp(timestamp_format="T")
print("Le timestamp avec l'heure seulement est {timestamp}".
↪format(timestamp=timestamp))
# Le timestamp avec l'heure seulement est 181434

timestamp = lz.get_timestamp(timestamp_format="NOW")
print("Le timestamp avec la date et l'heure sans retraitement est {timestamp}".
↪format(timestamp=timestamp))
# Le timestamp avec la date et l'heure sans retraitement est 2020-12-09 18:14:34

timestamp = lz.get_timestamp(timestamp_format="GD")
print("Le timestamp avec la date sans retraitement est {timestamp}".
↪format(timestamp=timestamp))
# Le timestamp avec la date sans retraitement est 2020-12-09

timestamp = lz.get_timestamp(timestamp_format="GT")
print("Le timestamp avec l'heure sans retraitement est {timestamp}".
↪format(timestamp=timestamp))
# Le timestamp avec l'heure sans retraitement est 18:14:34
```

Terminal :

```
$ python get_timestamp.py
Le timestamp sans milli secondes est 20201025_182141
Le timestamp en milli secondes est 457668
Le timestamp avec milli secondes est 20201025_182141.457668
Le timestamp avec la date seulement est 20201025
Le timestamp avec l'heure seulement est 182141
Le timestamp avec la date et l'heure sans retraitement est 2020-10-25 18:21:41
Le timestamp avec la date sans retraitement est 2020-10-25
Le timestamp avec l'heure sans retraitement est 18:21:41
```

corelibs.lazy.**get_username**()

Description

Récupère le nom de l'utilisateur courant

Returns

nom de l'utilisateur courant

Exemple :

```
# get_username.py
from corelibs import config, lazy as lz

username = lz.get_username()
print("Le nom de l'utilisateur actuel est \"{username}\"".
↪format(username=username))
```

corelibs.lazy.**is_file_exists**(*file_path*, *is_dir=False*, *continue_on_error=False*)

Description

Vérifie si le fichier existe ou non

Parameters

- **file_path** – chemin du fichier/répertoire
- **is_dir** – indique si la vérification porte sur un dossier ou non

valeurs possibles: *False/True*

valeur par défaut: *False* (la vérification se fait sans distinction)

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

True/False

Exemple :

```
# is_file_exists.py
from corelibs import lazy as lz


def test(file_test):
    if lz.is_file_exists(file_test):
        print("Le fichier \"{file_test}\" existe".format(file_test=file_test))
    else:
        print("Le fichier \"{file_test}\" n'existe pas".format(file_test=file_
↪test))


file = r"D:\OneDrive\Documents\_TEST_"
test(file)

file = r"D:\OneDrive\Documents\_TEST_\__LOGS__\Fichier.txt"
test(file)

file = r"D:\OneDrive\Documents\_TEST_"
if lz.is_file_exists(file, is_dir=True):
    print("Le fichier \"{file}\" existe et est un répertoire".format(file=file))

file = r"D:\OneDrive\Documents\_TEST_\Fichier.txt"
if lz.is_file_exists(file):
    print("Le fichier \"{file}\" existe".format(file=file))

file = r"D:\OneDrive\Documents\_TEST_\Fichier.txt"
if not lz.is_file_exists(file, is_dir=True):
    print("Le fichier \"{file}\" n'est pas un répertoire".format(file=file))
```

Terminal :

```
$ python get_abspath.py
Le fichier "D:\OneDrive\Documents\_TEST_\__LOGS__" existe
Le fichier "D:\OneDrive\Documents\_TEST_\__LOGS__\Fichier.txt" existe
Le fichier "D:\OneDrive\Documents\_TEST_\__LOGS__\Fichier.txt" existe et est un␣
↪répertoire
Le fichier "D:\OneDrive\Documents\_TEST_\__LOGS__\Fichier.txt" existe et n'est␣
↪pas un répertoire
```

corelibs.lazy.**is_jupyter**()

Description

Permet de savoir si l'environnement sur lequel est exécuté le script python est Jupyter Notebook ou un terminal QTConsole Jupyter

---

Returns

Booléen: *False/True*

Exemple :

```
# is_jupyter.py
from corelibs import lazy as lz

print(lz.is_jupyter())
```

corelibs.lazy.**is_namedtuple_instance**(*obj*)

Description

Permet de déterminer si l'objet passé en argument est une instance de tuple nommé.

Parameters **obj** – un tuple nommé

Returns

Booléen: *False/True*

Exemple :

```
# is_namedtuple_instance.py
from collections import namedtuple
from corelibs import lazy as lz


# création d'un objet PuPuce tuple nommé
PuPuce = namedtuple("PuPuce", ["name", "age"])
kim = PuPuce(
    name="Kim",
```

(continues on next page)

(continued from previous page)

```
    age=6
)
print(kim)  # affiche bien PuPuce(name='Kim', age=6)

print(lz.is_namedtuple_instance(kim))  # affiche bien True

print(lz.is_namedtuple_instance("Coucou Kim"))  # affiche False
```

corelibs.lazy.**is_platform**(*platform_os='Windows'*, *continue_on_error=False*)

Description

Permet de vérifier le système d'exploitation sur lequel est lancé le script python

---

Parameters

- **platform_os** – indique quel est la plateforme de référence à vérifier .

  valeurs possibles: "Windows", "Linux" ou "OSX"
  valeur par défaut: "Windows"

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

Booléen: *False/True*

Exemple :

```
# is_platform.py
from corelibs import config, lazy as lz

print(lz.is_platform("Windows"))  # vérfie si l'OS est Windows ou non...
print(lz.is_platform("Linux"))
print(lz.is_platform("OSX"))

print(lz.is_platform("INCONNUE"))
```

corelibs.lazy.**is_stdin**()

Description

Permet de savoir si l'environnement sur lequel est exécuté le script python est un terminal standard ou non

Returns

Booléen: *False/True*

corelibs.lazy.**is_validated_schema**(*data_2_validate*, *schema*, *is_dict_schema=True*, *verbose=False*, *continue_on_error=False*)

Description

Vérifie si une donnée est conforme au schéma descriptif. La donnée à éprouver est par défaut un dictionnaire autrement, c'est un fichier de configuration YAML.

Parameters

- **data_2_validate** – donnée à valider

  dictionnaire ou tuple (si is_dict_schema == True)
  ou
  chemin absolu du fichier YAML à valider (si is_dict_schema == False)

- **schema** – nom du schéma de référence

  schéma (si is_dict_schema == True)
  ou
  chemin absolu du schéma YAML permettant la validation (si is_dict_schema == False)

- **is_dict_schema** – indique si le schéma est un dictionnaire ou un fichier YAML

  valeurs possibles: *False/True*
  valeur par défaut: *True*

- **verbose** – afficher ou non les messages d'info/alertes

valeurs possibles: *False/True*
valeur par défaut: *DEFAULT_VERBOSE* (cf. *Configurations*)

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

Booléen: *False/True*

Exemple :

```
# is_validated_schema.py
import schema as sc

from corelibs import lazy as lz

########################################################################################
↪#####################################
# TEST Schéma DICT (défaut)
########################################################################################
↪#####################################
# Test schéma byte format size
SCHEMA_DEFAULT_BYTE_SIZE_FORMAT = sc.Schema({
    sc.Optional(sc.And(str, lambda s: s in (
        # clé parmi les noms définis ci-dessous
        "octet", "Ko", "Mo", "Go", "To"
    ))): {
        "min_size": sc.Or(int, float)
    }
})
# Test valide
DEFAULT_BYTE_SIZE_FORMAT_OK = {
    "octet": {"min_size": 0},
    "Ko": {"min_size": 1},
    "Mo": {"min_size": 1},
    "Go": {"min_size": 1},
    "To": {"min_size": 0.5}
}
lz.is_validated_schema(DEFAULT_BYTE_SIZE_FORMAT_OK, SCHEMA_DEFAULT_BYTE_SIZE_
↪FORMAT, verbose=True)
# Test invalide
DEFAULT_BYTE_SIZE_FORMAT_KO = {
    "octet": {"min_size": "Hello"},
    "Ko": {"min_size": "Kim"},
    "Mo": {"min_size": "Marie"},
    "Go": {"min_size": "Adélie"},
    "To": {"min_size": 7}
}
```

(continues on next page)

(continued from previous page)

```
lz.is_validated_schema(DEFAULT_BYTE_SIZE_FORMAT_KO, SCHEMA_DEFAULT_BYTE_SIZE_
↪FORMAT, verbose=True)
# Test schéma scaffolding
SCHEMA_DIR_SCAFFOLDING = sc.Schema({
    sc.Optional(sc.And(str, lambda s: s in (
        # clé parmi les noms définis ci-dessous
        "input", "output", "logs", "docs"
    ))): {
        "name": sc.Regex(
            r"^__[a-zA-Z0-9 _-]+__$"
        ),
        "make": bool
    }
})
# Test valide
DIR_SCAFFOLDING_OK = {
    "input": {  # dossier contenant toutes les données "entrées"
        "name": "__MY_INPUTS__",
        "make": True
    },
    "output": {  # dossier contenant toutes les données "sorties"
        "name": "__MY_OUTPUTS__",
        "make": True
    },
    "logs": {  # dossier contenant toutes les sorties "logs"
        "name": "__MY-LOGS__",
        "make": True
    },
}
lz.is_validated_schema(DIR_SCAFFOLDING_OK, SCHEMA_DIR_SCAFFOLDING, verbose=True)
# Test invalide
DIR_SCAFFOLDING_KO = {
    "input": {  # dossier contenant toutes les données "entrées"
        "name": "__R2 D2__",
        "make": False
    },
    "output": {  # dossier contenant toutes les données "sorties"
        "name": "__MY_OUTPUTS__",
        "make": "Coucou"
    },
    "logs": {  # dossier contenant toutes les sorties "logs"
        "name": "__MY-LOGS__",
        "make": True
    },
}
lz.is_validated_schema(DIR_SCAFFOLDING_KO, SCHEMA_DIR_SCAFFOLDING)
# Test schéma styles des champs
SCHEMA_FIELD_STYLES = sc.Schema({
    sc.Optional(sc.And(str, lambda s: s in (
        "asctime", "hostname", "username", "levelname", "name", "programname"
    ))): {
        "color": sc.Or(sc.And(int, lambda n: 0 <= n <= 255),
                       sc.And(str, lambda s: s in (
                           "black", "blue", "cyan", "green",
                           "magenta", "red", "white", "yellow"
                       ))),
        sc.Optional(sc.And(str, sc.Use(str.lower), lambda s: s in (
```

(continues on next page)

(continued from previous page)

```
            "bold", "bright", "faint"
        ))): bool,
    }
})
# Test valide...
FIELD_STYLES_OK = {
    "asctime": {"color": 242, "bright": True},
    "hostname": {"color": "magenta"},
    "username": {"color": "yellow"},
    "levelname": {"color": 242, "bright": True},
    "name": {"color": "blue"},
    "programname": {"color": "cyan"}
}
lz.is_validated_schema(FIELD_STYLES_OK, SCHEMA_FIELD_STYLES)
# Test invalide...
FIELD_STYLES_KO = {
    "asctimeZ": {"color": 242, "bright": True},
    "hostname": {"color": "white"},
    "username": {"color": "yellow"},
    "levelname": {"color": 242, "bright": False},
    "name": {"color": "blue"},
    "programname": {"color": "cyan"}
}
lz.is_validated_schema(FIELD_STYLES_KO, SCHEMA_FIELD_STYLES)
# Test schéma styles des niveaux d'alertes
SCHEMA_LEVEL_STYLES = sc.Schema({
    sc.Optional(sc.And(str, lambda s: s in (
        "critical", "error", "warning", "debug", "info", "notice", "spam",
↪"success", "verbose"
    ))): {
        "color": sc.Or(sc.And(int, lambda n: 0 <= n <= 255),
                       sc.And(str, lambda s: s in (
                           "black", "blue", "cyan", "green",
                           "magenta", "red", "white", "yellow"
                       ))),
        sc.Optional(sc.And(str, sc.Use(str.lower), lambda s: s in (
            "background"
        ))): sc.Or(sc.And(int, lambda n: 0 <= n <= 255),
                   sc.And(str, lambda s: s in (
                       "black", "blue", "cyan", "green",
                       "magenta", "red", "white", "yellow"
                   ))),
    }
})
# Test valide...
LEVEL_STYLES_OK = {
    "critical": {"color": "white", "background": "red"},
    "verbose": {"color": "white"},
}
lz.is_validated_schema(LEVEL_STYLES_OK, SCHEMA_LEVEL_STYLES)
# Test invalide...
LEVEL_STYLES_KO = {
    "WRONG_KEY_critical": {"color": "white", "background": "red"}
}
lz.is_validated_schema(LEVEL_STYLES_KO, SCHEMA_LEVEL_STYLES)
# Test schéma format d'affichage des logs
SCHEMA_LOG_FORMAT = sc.Regex(
```

(continues on next page)

(continued from previous page)

```
    r"^([ <>•@:=$~{}\(\)\[\]\w\d\-\.]*"
    + r"%\((\
↪b(asctime|created|filename|funcName|levelname|levelno|lineno|message|module|msecs|name
↪"
    + r
↪"|pathname|process|processName|relativeCreated|thread|threadName|username|hostname)\
↪b)\)"
    + r"\d*[sd]{1,1}[ <>•@:=$~{}\(\)\[\]\w\d\-\.]*)*$"
)
# Test valide...
LOG_FORMAT_OK = \
    "> %(asctime)s %(username)s@%(hostname)s - %(name)s" \
    + "[P.%(process)d - T.%(thread)d - L.%(lineno)05d] • %(levelname)13s
↪%(message)s"
lz.is_validated_schema(LOG_FORMAT_OK, SCHEMA_LOG_FORMAT)
# Test invalide...
LOG_FORMAT_KO = "%(asctime)s %(username)s@%(hostname)s %(name)s[%(process)d] • %(_
↪levelname_)13s %(message)s"
lz.is_validated_schema(LOG_FORMAT_KO, SCHEMA_LOG_FORMAT)
# Test schéma format d'affichage timestamp des logs
SCHEMA_LOG_DATE_FORMAT = sc.Regex(r"^(%(%|\b[aAwdbBmyYHIpMSfzZjUWcxXGuV]\b)[ \/\-
↪:]*)*$")
# Test valide...
LOG_DATE_FORMAT_OK = "%Y/%m/%d %H:%M:%S"
lz.is_validated_schema(LOG_DATE_FORMAT_OK, SCHEMA_LOG_DATE_FORMAT)
# Test invalide...
LOG_DATE_FORMAT_KO = "%Y-%m-%d %HH : %M : %S"
lz.is_validated_schema(LOG_DATE_FORMAT_KO, SCHEMA_LOG_DATE_FORMAT)


################################################################################
↪#####################################
# TEST Schéma YAML
################################################################################
↪#####################################
lz.is_validated_schema(
    r"D:\OneDrive\Documents\[PYTHON_PROJECTS]\corelibs\tests\lazy\conf_test.yaml",
    r"D:\OneDrive\Documents\[PYTHON_PROJECTS]\corelibs\tests\lazy\schema_conf_
↪test.yaml",
    is_dict_schema=False
)
```

Exemple Schéma YAML :

```
# schema_conf_test.yaml
list(include("person"), min=1)
---
person:
  first name: str()
  last name: str()
  age: int(max=130, required=False)
```

Exemple fichier YAML à valider avec le schéma descriptif YAML :

```
- first name: "Kim Marie Adélie"
  last name: "TRUONG"
  age: 6
```

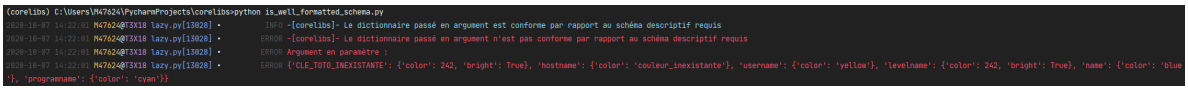
(continues on next page)

(continued from previous page)

```
- first name: "Anne"
  last name: "TRUONG"

- first name: "Michel"
  last name: "TRUONG"
  age: 231  # erreur car age supérieur à 130 qui est le maximum autorisé
  unknown key: "yeah!"  # erreur car clé inconnue

- mt: False  # erreur car clé inconnu dans un nouveau bloc
```

Terminal :

```
$ python is_validated_schema.py  #illustration indicative non nécessairement␣
↪représentative du code en exemple
```



corelibs.lazy.**merge_dictionaries**(*merged_dictionary*, *dictionary_2_merge*)

Description

Permet de fusionner 2 dictionnaires ensemble
/!\ ATTENTION /!\ L'ordre des dictionnaires en argument est important!

Parameters

- **merged_dictionary** – dictionnaire conteneur destiné à recevoir le résultat de la fusion
- **dictionary_2_merge** – dictionnaire à fusionner

Returns

dictionnaire résultat de la fusion
ou
None

Exemple :

```
# merge_dictionaries.py
from corelibs import lazy as lz

x = None
y = {"prenom": "Kim", "age": 6}
z = lz.merge_dictionaries(x, y)
print(z)  # {'prenom': 'Kim', 'age': 6}
```

(continues on next page)

(continued from previous page)

```
# inversion
z = lz.merge_dictionaries(y, x)
print(z)  # None

x = {"a": 1, "b": 2}
y = {"b": 10, "c": 11}
z = lz.merge_dictionaries(x, y)
print(z)  # {"a": 1, "b": 10, "c": 11}

x = {"b": 10, "c": 11}
y = {"a": 1, "b": {"b1": "hello", "b2": 3}}
z = lz.merge_dictionaries(x, y)
print(z)  # {"b": {"b1": "hello", "b2": 3}, "c": 11, "a": 1}

x = {"b": {"b1": "hello", "b2": 3}, "c": 11}
y = {"a": 1, "b": {"b1": "Kim", "b2": 6}}
z = lz.merge_dictionaries(x, y)
print(z)  # {"b": {"b1": "Kim", "b2": 6}, "c": 11, "a": 1}

DICT_USER = {
    "asctime": {"color": "BLACK"},
    "levelname": {"color": "WHITE"}
}

DICT_REF = {
    "asctime": {"color": 242, "bright": True},
    "hostname": {"color": "magenta"},
    "username": {"color": "yellow"},
    "levelname": {"color": 242, "bright": True},
    "name": {"color": "blue"},
    "programname": {"color": "cyan"}
}

MERGED_DICT = lz.merge_dictionaries(DICT_REF, DICT_USER)
print(MERGED_DICT)  # {
#   "asctime": {"color": "BLACK", "bright": True},
#   "hostname": {"color": "magenta"},
#   "username": {"color": "yellow"},
#   "levelname": {"color": "WHITE", "bright": True},
#   "name": {"color": "blue"},
#   "programname": {"color": "cyan"}
# }
```

corelibs.lazy.**mkdir**(*location=None*, *make_scaffolding=True*, *dir_scaffolding={'docs': {'make': True, 'name': '__DOCS__'}, 'input': {'make': True, 'name': '__INPUTS__'}, 'logs': {'make': True, 'name': '__LOGS__'}, 'output': {'make': True, 'name': '__OUTPUTS__'}}*, *verbose=False*, *continue_on_error=False*)

Description

Créer un répertoire standard ou une liste de répertoires de manière récursive ; si le ou les parents n'existent pas, ils seront créés (dans le cas d'une liste de répertoires cela permet de factoriser les instructions).

Par défaut, lorsque *corelibs.lazy.mkdir()* est appelé, 4 répertoires en plus sont créés :

- 1 pour recevoir les fichiers logs, nommé par défaut "__LOGS__"
- 1 pour recevoir les sorties, nommé par défaut "__OUTPUTS__"
- 1 pour recevoir les entrées, nommé par défaut "__INPUTS__"
- 1 pour recevoir les documentations et/ou spécifications, nommé par défaut "__DOCS__"

Note:

Les noms respectifs des répertoires modèles sont gérés par le dictionnaire
config.DEFAULT_DIR_SCAFFOLDING (cf. *Configurations* pour les détails.)

Il est possible de modifier les noms et/ou la création des répertoires modèles en chargeant un nouveau dictionnaire lors de l'appel de `corelibs.lazy.mkdir()`, directement en argument ou via un écrasement de la constante `DEFAULT_DIR_SCAFFOLDING`.

Parameters

- **`location`** – la location du répertoire à créer.

  valeurs possibles: chemin absolu ou tuple/tableau de chemins absolus
  valeur par défaut: le chemin retourné par `corelibs.lazy.get_module_path()`

- **`make_scaffolding`** – indique s'il faut ou non créer les dossiers "modèle"

  valeurs possibles: *Dictionnaire*
  valeur par défaut: *DEFAULT_DIR_SCAFFOLDING* (cf. *Configurations*)

- **`dir_scaffolding`** – dictionnaire définissant les noms des dossiers "modèle" et s'il faut ou non les créer unitairement

  valeurs possibles: *False/True*
  valeur par défaut: *True*

- **`verbose`** – afficher ou non les messages d'info/alertes

  valeurs possibles: *False/True*
  valeur par défaut: *DEFAULT_VERBOSE* (cf. *Configurations*)

- **`continue_on_error`** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*

valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

rien…

Exemple :

```
# mkdir.py
from corelibs import lazy as lz


# création par défaut où se trouve l'emplacement du programme `mkdir.py`
lz.mkdir()


# création d'un répertoire standard avec les parents si n'existe pas, sans la␣
↪structure "modèle"
lz.mkdir(
    location=r"D:\OneDrive\Documents\_TEST_\PARENTS\INEXISTANTS\DOSSIER_STANDARD",
    make_scaffolding=False,
    verbose=True
)


# création avec 4 dossiers par défaut de la structure "modèle"
# à l'emplacement "D:\OneDrive\Documents\_TEST_\PARENTS\INEXISTANTS\DOSSIER_
↪STANDARD"
lz.mkdir(location=r"D:\OneDrive\Documents\_TEST_\PARENTS\DOSSIERS_SCAFFOLD",␣
↪verbose=True)


# création personnalisée avec un seul dossier "__MY-LOGS__"
# à l'emplacement "D:\OneDrive\Documents\_TEST_\PARENTS\INEXISTANTS\DOSSIER_
↪STANDARD"
lz.mkdir(
    location=r"D:\OneDrive\Documents\_TEST_\PARENTS\INEXISTANTS\DOSSIER_STANDARD",
    dir_scaffolding={
        "input": {  # dossier contenant toutes les données "entrées"
            "name": "__R2 D2__",
            "make": False
        },
        "output": {  # dossier contenant toutes les données "sorties"
            "name": "__MY_OUTPUTS__",
            "make": False
        },
        "logs": {  # dossier contenant toutes les sorties "logs"
            "name": "__MY-LOGS__",
            "make": True
        },
        "docs": {  # dossier contenant toutes les documentations/specs liées au␣
↪projet
            "name": "__DOCS__",
            "make": False
```

(continues on next page)

(continued from previous page)

```
        },
    },
    verbose=True
)


# création d'une liste de répertoires standards (sans les répertoires modèles) ayant pour structure
# ...DOSSIER_T42020
#     |_Dossier A
#     |_Dossier B
#       |_SDossier B1
#       |_SDossier B2
lz.mkdir(
    location=(
        r"D:\OneDrive\Documents\_TEST_\DOSSIER_T42020\Dossier A",
        r"D:\OneDrive\Documents\_TEST_\DOSSIER_T42020\Dossier B\SDossier B1",
        r"D:\OneDrive\Documents\_TEST_\DOSSIER_T42020\Dossier B\SDossier B2",
    ),
    make_scaffolding=False,
    verbose=True
)  # dans la mesure où les dossiers parents sont créés si n'existent pas, une factorisation est possible...


# création d'une liste de répertoires standards (avec les répertoires modèles personnalisés) ayant pour structure
# ...DOSSIER_T12021
#     |_Dossier A
#       |__MY_INPUTS__
#       |__MY_OUTPUTS__
#       |__MY_LOGS__
#     |_Dossier B
#       |_SDossier B1
#         |_ +3 DOSSIERS MODELES PERSONNALISÉS
#       |_SDossier B2
#         |_ +3 DOSSIERS MODELES PERSONNALISÉS
lz.mkdir(
    location=(
        r"D:\OneDrive\Documents\_TEST_\DOSSIER_T12021\Dossier A",
        r"D:\OneDrive\Documents\_TEST_\DOSSIER_T12021\Dossier B\SDossier B1",
        r"D:\OneDrive\Documents\_TEST_\DOSSIER_T12021\Dossier B\SDossier B2",
    ),
    dir_scaffolding={
        "input": {  # dossier contenant toutes les données "entrées"
            "name": "__MY_INPUTS__",
            "make": True
        },
        "output": {  # dossier contenant toutes les données "sorties"
            "name": "__MY_OUTPUTS__",
            "make": True
        },
        "logs": {  # dossier contenant toutes les sorties "logs"
            "name": "__MY_LOGS__",
            "make": True
        },
        "docs": {  # dossier contenant toutes les documentations/specs liées au projet
```

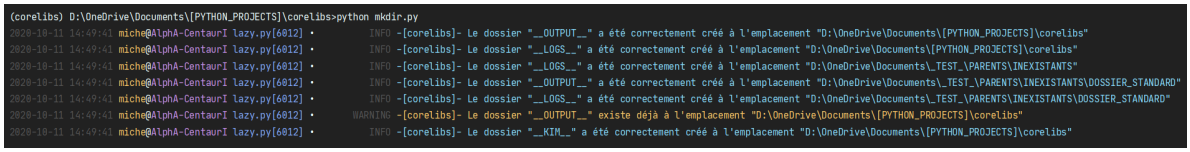
(continues on next page)

(continued from previous page)

```
            "name": "__DOCS__",
            "make": False
        },
    },
    verbose=True
)  # cette construction n'a aucun intérêt fonctionnel mais cela est possible...
```

Terminal :

```
$ python mkdir.py  #illustration indicative non nécessairement représentative du␣
↪code en exemple
```



corelibs.lazy.**move**(*source*, *destination*)

Description

Permet de déplacer un ou des fichiers vers une nouvelle destination ou des nouvelles destinations. Les fichiers sont au sens Unix du terme (i.e. soit fichier régulier, soit répertoire)

Parameters

- **source** – indique l'emplacement source du ou des fichiers à déplacer avec le(s) chemin(s) absolu(s), avec ou sans schéma.
- **destination** – indique l'emplacement destination du ou des fichiers à déplacer avec le(s) chemin(s) absolu(s).

Returns

rien...

Exemple :

```
# move.py
from corelibs import lazy as lz


# Création dossier destination
repertoire_destination = r"D:\OneDrive\Documents\_TEST_\_NEW_DESTINATION_"
lz.mkdir(repertoire_destination, make_scaffolding=False)

# Déplacement simple de fichier standard, sans renommage
lz.move(r"\\wsl$\Ubuntu-20.04\root\.zsh_history", repertoire_destination)  #␣
↪chemin réseau...
```

(continues on next page)

(continued from previous page)

```
# Déplacement simple de fichier standard, avec renommage
lz.move(r"D:\OneDrive\Documents\_TEST_\_éèçàoöôîïêëùûü;.txt", repertoire_
↪destination + "\\nouveau_nom.txt")

# Déplacement simple de répertoire standard, sans renommage
lz.move(r"D:\OneDrive\Documents\_TEST_\__R2D2-LOGS__", repertoire_destination)

# Déplacement simple de répertoire standard, avec renommage
lz.move(r"D:\OneDrive\Documents\_TEST_\__R2D2-LOGS__", repertoire_destination + "\
↪\__R2D2__")

# Déplacement via mode modèle
lz.move(r"D:\OneDrive\Documents\_TEST_\*.sas*", repertoire_destination)  # modèle␣
↪avec extension
# ou
lz.move(r"D:\OneDrive\Documents\_TEST_\*2020-11-11*", repertoire_destination)  #␣
↪modèle sans extension, comprenant la chaîne "2020-11-11" dans le nom

# Déplacement groupé dans un dossier
lz.move((
    r"D:\OneDrive\Documents\_TEST_\*.sas*",  # avec schéma
    r"D:\OneDrive\Documents\_TEST_\2020-11-11.jpg",
    r"D:\OneDrive\Documents\_TEST_\__R2D2-LOGS__"  # dossier...
), repertoire_destination)

# Déplacement groupé d'une liste de fichiers dans une autre liste de fichiers␣
↪(fonctionnement 1-1, i.e. même nombre de fichiers en entrée et en sortie,␣
↪traitée de manière itérative)
lz.move((
    r"D:\OneDrive\Documents\_TEST_\*.sas*",  # avec schéma
    r"D:\OneDrive\Documents\_TEST_\2020-11-11.jpg",
    r"D:\OneDrive\Documents\_TEST_\__R2D2-LOGS__",  # dossier...
), (
    repertoire_destination,  # sans renommage
    repertoire_destination + "\\2020-11-11_NOUVEAU_NOM.jpg",  # avec renommage
    repertoire_destination + "\\__R2D2-NEW__",  # avec renommage
))


############################################################################
↪#####################################
# NOTES ####################################################################
↪#####################################
# Comme sous Unix, move peut être utilisé pour renomer un fichier (chemin source␣
↪= chemin cible)
############################################################################
↪#####################################
lz.move(r"D:\OneDrive\Documents\_TEST_\_éèçàoöôîïêëùûü;.txt", r"D:\OneDrive\
↪Documents\_TEST_\NOUVEAU_NOM.txt")
```

corelibs.lazy.**open_explorer**(*path*)

Description

Permet d'ouvrir dans l'explorateur en pointant directement sur le chemin passé en argument.

Parameters **path** – indique le chemin à pointer

Returns

rien…

Exemple :

```
# open_explorer.py
from corelibs import lazy as lz

lz.open_explorer(lz.get_home())
lz.open_explorer(r"D:\OneDrive\Documents\_TEST_")

lz.open_explorer(lz.get_home() + r"\.corelibs")  # ouvre le dossier contenant les␣
↪données utilisateurs comme user_config.py
```

corelibs.lazy.**rename**(*path*, *pattern*, *replace*, *transform=None*, *debug=True*, *verbose=False*)

Description

Permet de renommer à la volée des fichiers selon des schémas regex.

Note:

Cette fonction est un emballage de la fonction *corelibs.lazy.move()*. A l'utiliser de préférence si c'est un renommage simple.

Warning:

Compte tenu des possibles dégâts liés aux erreurs de schéma regex, par défaut, la fonction s'exécute en mode debug = True

Pour que la fonction s'applique, il faut donc que le debug soit positionné à False **explicitement**.

Parameters

- **path** – indique l'emplacement des fichiers sources à renommer en chemin absolu.

- **pattern** – indique le schéma regex des fichiers sources. Le schéma source doit contenir à minima des (), listant les différents sous groupes. Un séquençace est possible également via les directives suivantes :
  - %{0} pour un séquençage sans padding
  - %{n} où n est un entier indiquant le nombre de 0 en padding
- **replace** – indique le schéma regex final de remplacement. Le schéma cible doit contenir à minima des \n où n est un entier représentant les sous groupes trouvés à partir du schéma source.
- **transform** – indique s'il y a des transformations à opérer ou non. Les transformations possibles sur le nom des fichiers cibles sont :
  - \U\n pour Uppercase sur le sous groupe identifié n
  - \L\n pour Lowercase sur le sous groupe identifié n
  - \C\n pour Capitalize (Premier mot en capitalizing) sur le sous groupe identifié n
  - \T\n pour Title (Premier Mot En Title) sur le sous groupe identifié n
  - \S\n pour Swapcase sur le sous groupe identifié n
- **debug** – indique si la fonction doit s'appliquer ou tourner à blanc

  valeurs possibles: *False/True*
  valeur par défaut: *True*

- **verbose** – afficher ou non les messages d'info/alertes. Dans le cadre d'une utilisation regex, le verbose peut se comporter comme un véritable spam... !!! =þ

  valeurs possibles: *False/True*
  valeur par défaut: *DEFAULT_VERBOSE* (cf. *Configurations*)

Returns

rien...

Exemple :

```
# rename.py
from corelibs import lazy as lz

# Renommage simple
lz.rename(
    path=r"D:\OneDrive\Documents\_TEST_\_TEST_RENOMMAGE_",
    pattern=r"_cars.sas7bdat",
    replace=r"nouveau_RS_cars.sas7bdat",
    debug=False
)  # équivalent à lz.move(r"D:\OneDrive\Documents\_TEST_\_TEST_RENOMMAGE_\_cars.
↪sas7bdat", r"D:\OneDrive\Documents\_TEST_\_TEST_RENOMMAGE_\nouveau_RS_cars.
↪sas7bdat")
```

(continues on next page)

(continued from previous page)

```
# Renommage simple par modèle
lz.rename(
    r"D:\OneDrive\Documents\_TEST_\_TEST_RENOMMAGE_\__R2D2-LOGS__",
    r"(termcolorlog)_(\d{4})(\d{2})(\d{2})_(\d{2})(\d{2})(\d{2})(.LOG)",  #␣
↪Schéma source avec 8 sous groupes, (termcolorlog)_(\d{4})(\d{2})(\d{2})_(\d{2}
↪)(\d{2})(\d{2})(.LOG)
    r"\1_\4\3\2_\5\6.log",  # Schéma cible décrivant "termcolorlog_JJMMAAAA_HHMM.
↪log"
    debug=False
)
# résultats :
# termcolorlog_20201208_222147.LOG -> termcolorlog_08122020_2221.log
# termcolorlog_20201208_222351.LOG -> termcolorlog_08122020_2223.log
# termcolorlog_20201208_222452.LOG -> termcolorlog_08122020_2224.log
# termcolorlog_20201208_222839.LOG -> termcolorlog_08122020_2228.log
# termcolorlog_20201208_223739.LOG -> termcolorlog_08122020_2237.log
# termcolorlog_20201208_223834.LOG -> termcolorlog_08122020_2238.log
# termcolorlog_20201208_223920.LOG -> termcolorlog_08122020_2239.log
# termcolorlog_20201208_224058.LOG -> termcolorlog_08122020_2240.log
# termcolorlog_20201208_224116.LOG -> termcolorlog_08122020_2241.log
# termcolorlog_20201208_224144.LOG -> termcolorlog_08122020_2241.log

# Renommage simple par modèle avec transformation 1er exemple
lz.rename(
    r"D:\OneDrive\Documents\_TEST_\_TEST_RENOMMAGE_\__R2D2-LOGS__\__R2D2-LOGS__",
    r"(termcolorlog)_(\d{4})(\d{2})(\d{2})_(\d{2})(\d{2})(\d{2})(.LOG)",  #␣
↪Schéma source avec 8 sous groupes, (termcolorlog)_(\d{4})(\d{2})(\d{2})_(\d{2}
↪)(\d{2})(\d{2})(.LOG)
    r"\1_\4\3\2_\5\6\8",  # Schéma cible décrivant "termcolorlog_JJMMAAAA_HHMM(.
↪LOG)"
    transform=r"\U1\T8",  # Upper sur le premier groupe et Title sur le dernier␣
↪groupe
    debug=False
)
# résultats :
# termcolorlog_20201208_222147.LOG -> TERMCOLORLOG_08122020_2221.Log
# termcolorlog_20201208_222351.LOG -> TERMCOLORLOG_08122020_2223.Log
# termcolorlog_20201208_222452.LOG -> TERMCOLORLOG_08122020_2224.Log
# termcolorlog_20201208_222839.LOG -> TERMCOLORLOG_08122020_2228.Log
# termcolorlog_20201208_223739.LOG -> TERMCOLORLOG_08122020_2237.Log
# termcolorlog_20201208_223834.LOG -> TERMCOLORLOG_08122020_2238.Log
# termcolorlog_20201208_223920.LOG -> TERMCOLORLOG_08122020_2239.Log
# termcolorlog_20201208_224058.LOG -> TERMCOLORLOG_08122020_2240.Log
# termcolorlog_20201208_224116.LOG -> TERMCOLORLOG_08122020_2241.Log
# termcolorlog_20201208_224144.LOG -> TERMCOLORLOG_08122020_2241.Log


# Renommage simple par modèle avec transformation 2ème exemple
lz.rename(
    r"D:\OneDrive\Documents\_TEST_\_TEST_RENOMMAGE_",
    r"_(cars)(.*)(.sas7bdat)",  # Schéma source avec 3 sous groupes
    r"NOUVEAU_NOM_\1_AVEC_TRANSFO\2.nouvelle_extension",  # Schéma cible␣
↪décrivant "NOUVEAU_NOM_(cars)_AVEC_TRANSFO(.*).nouvelle_extension"
    transform=r"\S1\T2",  # Swapcase sur le premier groupe et Title sur le 2ème␣
↪groupe
    debug=False
```

(continues on next page)

(continued from previous page)

```
)
# résultats :
# _CaRS.sas7bdat -> NOUVEAU_NOM_cArs_AVEC_TRANSFO.nouvelle_extension
# _cars_asia.sas7bdat -> NOUVEAU_NOM_CARS_AVEC_TRANSFO_Asia.nouvelle_extension


# Renommage simple par modèle avec transformation et séquences sans padding 1er␣
↪exemple
lz.rename(
    r"D:\OneDrive\Documents\_TEST_\_TEST_RENOMMAGE_\__R2D2__",
    r"(termcolorlog)_(\d{4})(\d{2})(\d{2})_(\d{2})(\d{2})(\d{2})(.LOG)",  #␣
↪Schéma source avec 8 sous groupes, (termcolorlog)_(\d{4})(\d{2})(\d{2})_(\d{2}
↪)(\d{2})(\d{2})(.LOG)
    r"\1_\4\3\2_%{0}_\8",  # Schéma cible décrivant "TERMCOLORLOG_JJMMAAAA_
↪%SEQUENCE_(.LOG)"
    transform=r"\U1\T8",  # Upper sur le premier groupe et Title sur le dernier␣
↪groupe
    debug=False,
    verbose=True
)
# résultats :
# termcolorlog_20201208_222147.LOG -> TERMCOLORLOG_08122020_1_.Log
# termcolorlog_20201208_222351.LOG -> TERMCOLORLOG_08122020_2_.Log
# termcolorlog_20201208_222452.LOG -> TERMCOLORLOG_08122020_3_.Log
# termcolorlog_20201208_222839.LOG -> TERMCOLORLOG_08122020_4_.Log
# termcolorlog_20201208_223739.LOG -> TERMCOLORLOG_08122020_5_.Log
# termcolorlog_20201208_223834.LOG -> TERMCOLORLOG_08122020_6_.Log
# termcolorlog_20201208_223920.LOG -> TERMCOLORLOG_08122020_7_.Log
# termcolorlog_20201208_224058.LOG -> TERMCOLORLOG_08122020_8_.Log
# termcolorlog_20201208_224116.LOG -> TERMCOLORLOG_08122020_9_.Log
# termcolorlog_20201208_224144.LOG -> TERMCOLORLOG_08122020_10_.Log

# Renommage simple par modèle avec transformation et séquences avec padding 2ème␣
↪exemple
lz.rename(
    r"D:\OneDrive\Documents\_TEST_\_TEST_RENOMMAGE_\__R2D2__\__R2D2-LOGS__",
    r"(termcolorlog)_(\d{4})(\d{2})(\d{2})_(\d{2})(\d{2})(\d{2})(.LOG)",  #␣
↪Schéma source avec 8 sous groupes, (termcolorlog)_(\d{4})(\d{2})(\d{2})_(\d{2}
↪)(\d{2})(\d{2})(.LOG)
    r"_%{10}_\1_\4\3\2_%{0}_\8",  # Schéma cible décrivant "_%SEQUENCE_PADDING10_
↪TERMCOLORLOG_JJMMAAAA_%SEQUENCE_PADDING_(.LOG)"
    transform=r"\U1\T8",  # Upper sur le premier groupe et Title sur le dernier␣
↪groupe
    debug=False
)
# résultats :
# termcolorlog_20201208_182147.LOG -> _0000000001_TERMCOLORLOG_08122020_
↪0000000001_.Log
# termcolorlog_20201208_182351.LOG -> _0000000002_TERMCOLORLOG_08122020_
↪0000000002_.Log
# termcolorlog_20201208_182452.LOG -> _0000000003_TERMCOLORLOG_08122020_
↪0000000003_.Log
# termcolorlog_20201208_182839.LOG -> _0000000004_TERMCOLORLOG_08122020_
↪0000000004_.Log
# termcolorlog_20201208_183739.LOG -> _0000000005_TERMCOLORLOG_08122020_
↪0000000005_.Log
# termcolorlog_20201208_183834.LOG -> _0000000006_TERMCOLORLOG_08122020_
↪0000000006_.Log
```

(continues on next page)

(continued from previous page)

```
# termcolorlog_20201208_183920.LOG -> _0000000007_TERMCOLORLOG_08122020_
↪0000000007_.Log
# termcolorlog_20201208_184058.LOG -> _0000000008_TERMCOLORLOG_08122020_
↪0000000008_.Log
# termcolorlog_20201208_184116.LOG -> _0000000009_TERMCOLORLOG_08122020_
↪0000000009_.Log
# termcolorlog_20201208_184144.LOG -> _0000000010_TERMCOLORLOG_08122020_
↪0000000010_.Log
```

corelibs.lazy.**reverse_named_tuple**(*named_tuple*, *convert_2_dict=False*, *continue_on_error=False*)

Description

Permet d'inverser l'ordre d'un tuple nommé.

Parameters

- **named_tuple** – indique le tuple nommé à inverser
- **convert_2_dict** – indique s'il faut convertir ou non le tuple nommé en dictionnaire

  valeurs possibles: *False/True*
  valeur par défaut: *False*

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

tuple nommé avec le nom de classe et attributs tels que passés en argument
ou
dictionnaire (conversion du tuple nommé)

Exemple :

```
# reverse_named_tuple.py
from corelibs import lazy as lz


# création d'un tuple nommé
```

(continues on next page)

(continued from previous page)

```
byte_size = lz.get_bytes_size_formats(1739886085)
print(byte_size)
# ByteSize(byte=1739886085, kilobyte=1699107.5, megabyte=1659.28, gigabyte=1.62,␣
↪terabyte=0.0)

# renverser le tuple nommé
reverse_nt = lz.reverse_named_tuple(byte_size)
print(reverse_nt)
# ByteSize(terabyte=0.0, gigabyte=1.62, megabyte=1659.28, kilobyte=1699107.5,␣
↪byte=1739886085.0)

# comme la classe originale existe lors de l'inversion, il est toujours possible d
↪'accéder à un des attributs
print(reverse_nt.megabyte)  # affichera 1659.28

# renverser le tuple nommé avec conversion en dictionnaire
reverse_nt = lz.reverse_named_tuple(byte_size, convert_2_dict=True)
print(reverse_nt)
# {'terabyte': 0.0, 'gigabyte': 1.62, 'megabyte': 1659.28, 'kilobyte': 1699107.5,
↪'byte': 1739886085}


# exemple avec dir_n_basename()
dir_n_basename = lz.get_dir_n_basename(r"C:\Users\M47624\corelibs\tests\lazy\get_
↪dir_n_basename.py")
print(dir_n_basename)
# DirPathnBaseName(dir_path='C:\\Users\\M47624\\corelibs\\tests\\lazy', base_name=
↪'get_dir_n_basename.py')

print(lz.reverse_named_tuple(dir_n_basename))
# DirPathnBaseName(base_name='get_dir_n_basename.py', dir_path='C:\\Users\\M47624\
↪\corelibs\\tests\\lazy')

print(dir_n_basename.base_name)  # get_dir_n_basename.py
```

CHAPTER

FOUR

MODULE LOG

Description générale

Module permettant de manipuler tout ce qui est relatif à la gestion des logs

class corelibs.log.**ColorLog**(*name=None*, *log_level=20*, *log_file_output=True*, *location=None*, *log_file_name=None*, *field_styles=None*, *level_styles=None*, *log_format=None*, *log_date_format=None*)

Description

Classe de base pour manipuler les logs colorées sans distinctions en sortie terminal ou Jupyter Notebooks.

cf. *Configurations* pour la configuration par défaut (modifiable lors de l'instanciation de la classe)

Note: ColorLog instancie dynamiquement la classe TermColorLog ou JupyterColorLog

pour plus de détails concernant les arguments :

- cf. *JupyterColorLog*
- cf. *TermColorLog*

Parameters

- **name** – indique le nom de la log en cours

  valeur possibles: *None/nom de la log*
  valeur par défaut: *None*

- **log_level** – indique le niveau minimum d'alerte pour l'affichage des logs

  valeur possibles: *None/niveau d'alerte*
  valeur par défaut: *config.DEFAULT_LOG_LEVEL*

- **log_file_output** – indique s'il faut ou non écrire les logs dans un fichier en sortie

  valeur possibles: *False/True*
  valeur par défaut: *True*

- **location** – indique l'emplacement des fichiers logs de sortie

  valeur possibles: *None/chemin dossier logs*
  valeur par défaut: *None*

  - **Si *location* non renseigné, par défaut, les logs seront enregistrés dans le dossier retourné par** *corelibs.lazy.mkdir()*
  - Sinon les logs seront enregistrés dans le dossier *location\DEFAULT_DIR_SCAFFOLDING["logs"]["name"]*

- **log_file_name** – indique le nom de la log

  valeur possibles: *None/nom de la log*
  valeur par défaut: *None*

  - **Si *log_file_name* non renseigné, par défaut, le nom sera la concaténation des retours de :**

    * *corelibs.lazy.get_module_name()*
    * *corelibs.lazy.get_file_extension()*
    * *corelibs.lazy.get_timestamp()*

    i.e. nom_programme_AAAAMMDD_HHMMDD.log, résultat de corelibs.lazy.get_file_extension(corelibs.lazy.get_module_name())[0] + "_" + corelibs.lazy.get_timestamp() + ".log"

- **field_styles** – permet de définir le style d'affichage des logs dans la sortie standard

  valeur possibles: *None/dictionnaire*
  valeur par défaut: *None*

- **level_styles** – permet de définir le style d'affichage des niveaux d'alerte des logs dans la sortie standard

  valeur possibles: *None/dictionnaire*
  valeur par défaut: *None*

- **log_format** – permet de définir le format d'affichage de la log

valeur possibles: *None/format d'affichage de la log*
valeur par défaut: *None*

- **log_date_format** – permet de définir le format d'affichage horodaté de la log

  valeur possibles: *None/format d'affichage horodaté de la log*
  valeur par défaut: *None*

Note:

cf. ***Configurations*** **:**

- *DEFAULT_LOG_LEVEL*
- *DEFAULT_FIELD_STYLES*
- *DEFAULT_LEVEL_STYLES*
- *DEFAULT_DIR_SCAFFOLDING*
- *DEFAULT_LOGS_EXTENSION*
- *DEFAULT_LOG_FORMAT*
- *DEFAULT_LOG_DATE_FORMAT*

Exemple :

```
# termcolorlog.py
from corelibs import config, log

# écrasement de la config par défaut de corelibs
# config.DEFAULT_VERBOSE = True
config.DEFAULT_LOGS_EXTENSION = ".LOG"
# config.DEFAULT_SHORT_LOG_LABEL = False
# config.DEFAULT_STACK_TRACE_2_FILE = True
# config.DEFAULT_STYLE_STACK_TRACE = "plaintext"
# config.DEFAULT_STACK_TRACE = True
# config.DEFAULT_CONTEXT_SOURCE_LINES = 7

# instanciation par défaut pour la sortie standard terminal seulement
cl = log.TermColorLog()

cl.debug("Bonjour, ceci est un test niveau DEBUG")
cl.info("Bonjour, ceci est un test niveau INFO", True)
cl.warning("Bonjour, ceci est un test niveau WARNING")
cl.error("Bonjour, ceci est un test niveau ERROR", trace_back=True)
cl.critical("Bonjour, ceci est un test niveau CRITIQUE")


# redéfinition du nom du dossier logs
# /!\ IMPORTANT /!\
#    il n'y a que le dossier des logs qui est pris en compte, définir les autres␣
↪dossiers ici est inutile
config.DEFAULT_DIR_SCAFFOLDING = {
```

(continues on next page)

(continued from previous page)

```
    "logs": {  # dossier contenant toutes les sorties "logs"
        "name": "__R2D2-LOGS__",
        "make": True
    },
}

# instanciation personnalisée
user_cl = log.TermColorLog(
    name="Ma log à moi que j'ai",
    log_level=None,
    log_file_output=True,
    location=r"D:\OneDrive\Documents\_TEST_\T32020",
    log_file_name=None,
    field_styles={
        "asctime": {"color": "black"},
        "name": {"color": "green"},
    },
    level_styles={
        "info": {
            "color": "white",
            "background": "cyan"
        },
    },
    log_format="%(asctime)s - %(name)s [%(filename)s:%(lineno)07d] <>
↪%(levelname)13s : %(message)s",
    log_date_format="%A %d %B %Y"
)

user_cl.debug("Bonjour, ceci est un test personnalisé niveau DEBUG")
user_cl.info("Bonjour, ceci est un test personnalisé niveau INFO")
user_cl.warning("Bonjour, ceci est un test personnalisé niveau WARNING")
user_cl.error("Bonjour, ceci est un test personnalisé niveau ERROR", trace_
↪back=True)
user_cl.critical("Bonjour, ceci est un test personnalisé niveau CRITIQUE", trace_
↪back=True)


# Création log dynamiquement selon contexte terminal standard ou Jupyter Notebooks
xcl = log.ColorLog(log_file_name="KIM")
xcl.debug("Bonjour, ceci est un test dynamique niveau DEBUG")
xcl.info("Bonjour, ceci est un test dynamique niveau INFO")
xcl.warning("Bonjour, ceci est un test dynamique niveau WARNING")
xcl.error("Bonjour, ceci est un test dynamique niveau ERROR", trace_back=True)
xcl.critical("Bonjour, ceci est un test dynamique niveau CRITIQUE", trace_
↪back=True)
```

class corelibs.log.**ColoredFormatter**(**args*, *colors: Optional[Dict[str, str]] = None*, ***kwargs*)

Initialize the formatter with specified format strings.

Initialize the formatter either with the specified format string, or a default as described above. Allow for specialized date formatting with the optional datefmt argument. If datefmt is omitted, you get an ISO8601-like (or RFC 3339-like) format.

Use a style parameter of '%', '{' or '$' to specify that you want to use one of %-formatting, `str.format()` (`{}`) formatting or `string.Template` formatting in your format string.

Changed in version 3.2: Added the `style` parameter.

format(*record*) → str
Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

class corelibs.log.**JupyterColorLog**(*log_level=20*)

Description

Classe de base pour manipuler les logs colorées sur les sorties web Jupyter Notebooks.

cf. *Configurations* pour la configuration par défaut (modifiable lors de l'instanciation de la classe)

Parameters **log_level** – indique le niveau minimum d'alerte pour l'affichage des logs

valeur possibles: *None/niveau d'alerte*
valeur par défaut: *config.DEFAULT_LOG_LEVEL*

Note:

cf. *Configurations* :

- *DEFAULT_LOG_LEVEL*

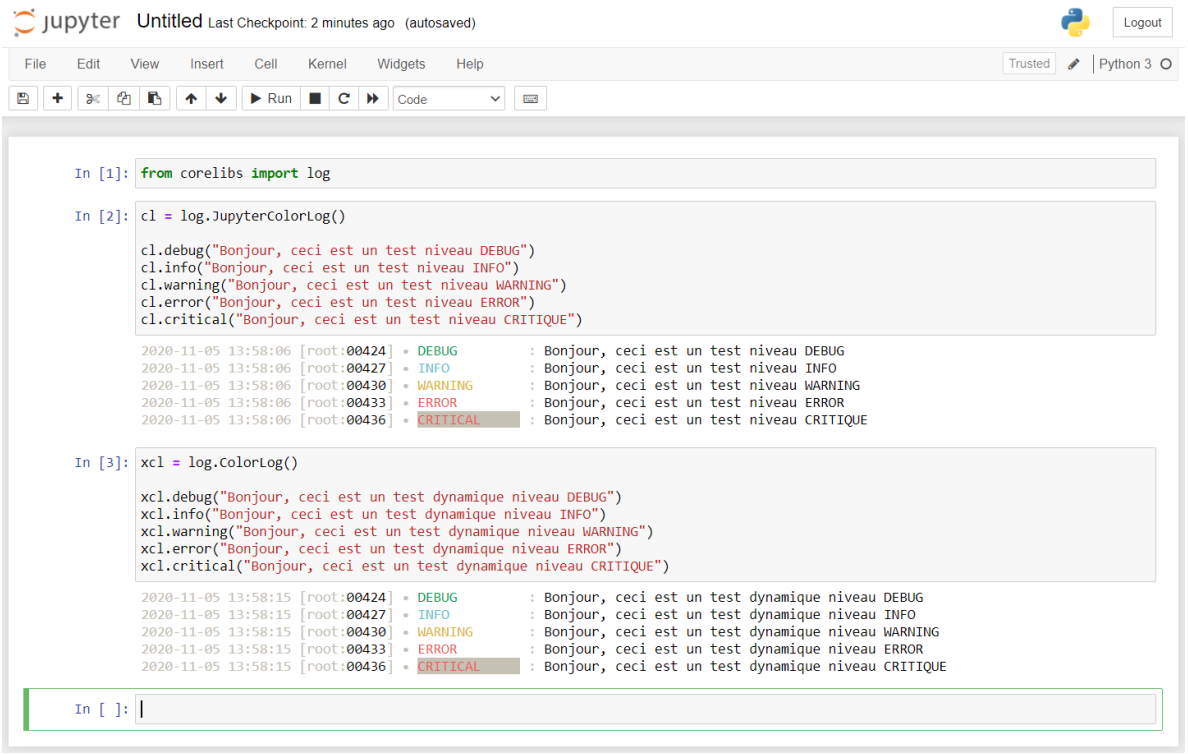
Exemple :

```
# jupytercolorlog.py
from corelibs import config, log

# écrasement de la config par défaut de corelibs
# config.DEFAULT_VERBOSE = True
# config.DEFAULT_STACK_TRACE = True
# config.DEFAULT_STYLE_STACK_TRACE = "color"
# config.DEFAULT_CONTEXT_SOURCE_LINES = 7

# instanciation par défaut pour Jupyter Notebooks seuleemnt
cl = log.JupyterColorLog()

cl.debug("Bonjour, ceci est un test niveau DEBUG")
cl.info("Bonjour, ceci est un test niveau INFO", trace_back=True)
cl.warning("Bonjour, ceci est un test niveau WARNING")
cl.error("Bonjour, ceci est un test niveau ERROR")
cl.critical("Bonjour, ceci est un test niveau CRITIQUE")
```

(continues on next page)

(continued from previous page)

```
# Création log dynamiquement selon contexte terminal standard ou Jupyter Notebooks
xcl = log.ColorLog()

xcl.debug("Bonjour, ceci est un test dynamique niveau DEBUG")
xcl.info("Bonjour, ceci est un test dynamique niveau INFO")
xcl.warning("Bonjour, ceci est un test dynamique niveau WARNING")
xcl.error("Bonjour, ceci est un test dynamique niveau ERROR")
xcl.critical("Bonjour, ceci est un test dynamique niveau CRITIQUE")
```



class corelibs.log.**StopWatch**(*log_handler=None*, *display_status_bar=True*)

Description

Classe permettant de chronométrer un programme ou une portion de programme

Le niveau minimal pour afficher les informations doit être de `log.INFO` (cf. *Configurations* pour plus d'informations *config.DEFAULT_LOG_LEVEL*)

Parameters

- **log_handler** – indique si le timing doit faire une sortie dans un fichier log, défini via l'instanciation de cf. *ColorLog* ou cf. *TermColorLog*.
- **display_status_bar** – indique s'il faut ou non afficher la barre de statut dans la sortie standard

valeur possibles: *False/True*
valeur par défaut: *True*

Returns

rien...

Exemple :

```
# stopwatch.py
from corelibs import log


# Instanciation log
cl = log.ColorLog()

# Instanciation Chronomètre programme
sw = log.StopWatch(cl)


# Ajout décorateur timing() pour calculer le temps d'exécution de la fonction `ma_
↪fonction()`
@log.timing()
# Cumuler décorateur status_bar() pour afficher la barre de statut
@log.status_bar()
def ma_fonction(nb):
    for _ in range(nb):
        pass


ma_fonction(200000000)
ma_fonction(200000000)

sw.stop()
```

stop()

Description

méthode pour arrêter le chronomètre

Returns

le temps total écoulé

class corelibs.log.**TermColorLog**(*name=None*, *log_level=20*, *log_file_output=True*, *location=None*, *log_file_name=None*, *field_styles=None*, *level_styles=None*, *log_format=None*, *log_date_format=None*)

Description

Classe de base pour manipuler les logs colorées dans la sortie standard du terminal.

cf. *Configurations* pour la configuration par défaut (modifiable lors de l'instanciation de la classe)

Parameters

- **name** – indique le nom de la log en cours

  valeur possibles: *None/nom de la log*
  valeur par défaut: *None*

- **log_level** – indique le niveau minimum d'alerte pour l'affichage des logs

  valeur possibles: *None/niveau d'alerte*
  valeur par défaut: *config.DEFAULT_LOG_LEVEL*

- **log_file_output** – indique s'il faut ou non écrire les logs dans un fichier en sortie

  valeur possibles: *False/True*
  valeur par défaut: *True*

- **location** – indique l'emplacement des fichiers logs de sortie

  valeur possibles: *None/chemin dossier logs*
  valeur par défaut: *None*

  – **Si *location* non renseigné, par défaut, les logs seront enregistrés dans le dossier retourné par** *corelibs.lazy.mkdir()*

  - Sinon les logs seront enregistrés dans le dossier *location\DEFAULT_DIR_SCAFFOLDING["logs"]["name"]*

- **log_file_name** – indique le nom de la log

  valeur possibles: *None/nom de la log*
  valeur par défaut: *None*

  - **Si *log_file_name* non renseigné, par défaut, le nom sera la concaténation des retours de :**

    * *corelibs.lazy.get_module_name()*
    * *corelibs.lazy.get_file_extension()*
    * *corelibs.lazy.get_timestamp()*

    i.e. nom_programme_AAAAMMDD_HHMMDD.log, résultat de corelibs.lazy.get_file_extension(corelibs.lazy.get_module_name())[0] + "_" + corelibs.lazy.get_timestamp() + ".log"

- **field_styles** – permet de définir le style d'affichage des logs dans la sortie standard

  valeur possibles: *None/dictionnaire*
  valeur par défaut: *None*

- **level_styles** – permet de définir le style d'affichage des niveaux d'alerte des logs dans la sortie standard

  valeur possibles: *None/dictionnaire*
  valeur par défaut: *None*

- **log_format** – permet de définir le format d'affichage de la log

  valeur possibles: *None/format d'affichage de la log*
  valeur par défaut: *None*

- **log_date_format** – permet de définir le format d'affichage horodaté de la log

  valeur possibles: *None/format d'affichage horodaté de la log*
  valeur par défaut: *None*

Note:

cf. ***Configurations*** **:**

- *DEFAULT_LOG_LEVEL*
- *DEFAULT_FIELD_STYLES*
- *DEFAULT_LEVEL_STYLES*
- *DEFAULT_DIR_SCAFFOLDING*
- *DEFAULT_LOGS_EXTENSION*
- *DEFAULT_LOG_FORMAT*
- *DEFAULT_LOG_DATE_FORMAT*

Exemple :

```
# termcolorlog.py
from corelibs import config, log

# écrasement de la config par défaut de corelibs
# config.DEFAULT_VERBOSE = True
config.DEFAULT_LOGS_EXTENSION = ".LOG"
# config.DEFAULT_SHORT_LOG_LABEL = False
# config.DEFAULT_STACK_TRACE_2_FILE = True
# config.DEFAULT_STYLE_STACK_TRACE = "plaintext"
# config.DEFAULT_STACK_TRACE = True
# config.DEFAULT_CONTEXT_SOURCE_LINES = 7

# instanciation par défaut pour la sortie standard terminal seulement
cl = log.TermColorLog()

cl.debug("Bonjour, ceci est un test niveau DEBUG")
cl.info("Bonjour, ceci est un test niveau INFO", True)
cl.warning("Bonjour, ceci est un test niveau WARNING")
cl.error("Bonjour, ceci est un test niveau ERROR", trace_back=True)
cl.critical("Bonjour, ceci est un test niveau CRITIQUE")


# redéfinition du nom du dossier logs
# /!\ IMPORTANT /!\
#    il n'y a que le dossier des logs qui est pris en compte, définir les autres␣
↪dossiers ici est inutile
config.DEFAULT_DIR_SCAFFOLDING = {
    "logs": {  # dossier contenant toutes les sorties "logs"
        "name": "__R2D2-LOGS__",
        "make": True
    },
}

# instanciation personnalisée
user_cl = log.TermColorLog(
    name="Ma log à moi que j'ai",
    log_level=None,
    log_file_output=True,
    location=r"D:\OneDrive\Documents\_TEST_\T32020",
    log_file_name=None,
    field_styles={
        "asctime": {"color": "black"},
        "name": {"color": "green"},
```

(continues on next page)

(continued from previous page)

```
    },
    level_styles={
        "info": {
            "color": "white",
            "background": "cyan"
        },
    },
    log_format="%(asctime)s - %(name)s [%(filename)s:%(lineno)07d] <>
↪%(levelname)13s : %(message)s",
    log_date_format="%A %d %B %Y"
)

user_cl.debug("Bonjour, ceci est un test personnalisé niveau DEBUG")
user_cl.info("Bonjour, ceci est un test personnalisé niveau INFO")
user_cl.warning("Bonjour, ceci est un test personnalisé niveau WARNING")
user_cl.error("Bonjour, ceci est un test personnalisé niveau ERROR", trace_
↪back=True)
user_cl.critical("Bonjour, ceci est un test personnalisé niveau CRITIQUE", trace_
↪back=True)


# Création log dynamiquement selon contexte terminal standard ou Jupyter Notebooks
xcl = log.ColorLog(log_file_name="KIM")
xcl.debug("Bonjour, ceci est un test dynamique niveau DEBUG")
xcl.info("Bonjour, ceci est un test dynamique niveau INFO")
xcl.warning("Bonjour, ceci est un test dynamique niveau WARNING")
xcl.error("Bonjour, ceci est un test dynamique niveau ERROR", trace_back=True)
xcl.critical("Bonjour, ceci est un test dynamique niveau CRITIQUE", trace_
↪back=True)
```

corelibs.log.**args_dumping**(*wrapped_func*)

Description

Décorateur pour lister tous les arguments passés dans une fonction décorée

Le niveau minimal pour afficher les informations doit être de `log.DEBUG` (cf. *Configurations* pour plus d'informations *config.DEFAULT_LOG_LEVEL*)

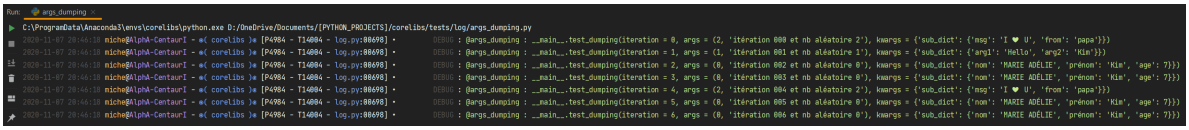
Exemple :

```
# args_dumping.py
from corelibs import config, log
from random import randrange

my_dict = {
    0: {
        "nom": "MARIE ADÉLIE",
        "prénom": "Kim",
        "age": 7
    },
```

(continues on next page)

(continued from previous page)

```
    1: {
        "arg1": "Hello",
        "arg2": "Kim"
    },
    2: {
        "msg": "I <3 U",
        "from": "papa"
    }
}


# le niveau d'alerte par défaut est à INFO, correspondant à la valeur 20...
# 10 correspond à DEBUG, ce qui a pour effet d'afficher le dumping...
config.DEFAULT_LOG_LEVEL = 10  # décommenter pour baisser le niveau d'alerte à␣
↪DEBUG


# décoration pour lister dynamiquement les arguments passés en paramétrage de la␣
↪fonction `test_dumping(...)
@log.args_dumping
def test_dumping(iteration, *args, **kwargs):
    pass


total_dict = len(my_dict)
for i in range(7):  # appel dynamique aléatoire d'arguments dans la fonction␣
↪`test_dumping()`
    rand_number = randrange(total_dict)
    test_dumping(
        i,
        rand_number,
        "itération {i} et nb aléatoire {rand_number}".format(i="{:0>3}".format(i),
↪ rand_number=rand_number),
        sub_dict=my_dict[rand_number]
    )
```



corelibs.log.**dict_dumping**(*wrapped_func*)

Description

Décorateur pour afficher toutes les fonctions qui retournent un dictionnaire dans un but de vérification.

Le niveau minimal pour afficher les informations doit être de `log.DEBUG` (cf. *Configurations* pour plus d'informations *config.DEFAULT_LOG_LEVEL*)

Exemple :

```python
# dict_dumping.py
from corelibs import config, log, tools as to

# le niveau d'alerte par défaut est à INFO, correspondant à la valeur 20...
# 10 correspond à DEBUG, ce qui a pour effet d'afficher le dumping...
config.DEFAULT_LOG_LEVEL = 10  # décommenter pour baisser le niveau d'alerte à␣
↪DEBUG


@log.dict_dumping
def test_namedtuple():
    filename = r"D:\OneDrive\Documents\_TEST_\2020-11-11.jpg"
    file_properties = to.get_file_properties(filename, pretty_byte_size=False)
    return file_properties


# affichage simple du tuple nommé pas toujours évident à lire selon complexcité
print(test_namedtuple())  # FileProperties(st_mode=33206, st_
↪ino=11540474045256220, st_dev=3199390331, st_nlink=1, st_uid='Invités', st_
↪gid=0, st_size=ByteSize(byte=98569, kilobyte=96.26, megabyte=0.09, gigabyte=0.0,
↪ terabyte=0.0), st_atime='14/11/2020 22:27:01', st_mtime='11/11/2020 22:10:46',␣
↪st_ctime='11/11/2020 22:10:39')
test_namedtuple()
```

```
Run:  dict_dumping
2020-11-18 22:31:50 miche@AlphA-CentaurI - ◉( corelibs )◉ [P9512 - T9688 - log.py:00759] •        DEBUG : Tuple détecté
st_atime: 14/11/2020 22:27:01
st_ctime: 11/11/2020 22:10:39
st_dev: 3199390331
st_gid: 0
st_ino: 11540474045256220
st_mode: 33206
st_mtime: 11/11/2020 22:10:46
st_nlink: 1
st_size:
  byte: 98569
  gigabyte: 0.0
  kilobyte: 96.26
  megabyte: 0.09
  terabyte: 0.0
st_uid: Invités


Process finished with exit code 0
```

corelibs.log.**stack_trace**(*force=False*)

Description

Décorateur pour afficher le détail des piles d'exécution (dans le cadre d'un débug).

Parameters **force** – permet de forcer localement l'affichage détaillé

cf. *Configurations* **:**

- *DEFAULT_STACK_TRACE*
- *DEFAULT_CONTEXT_SOURCE_LINES*

Returns

rien...

Exemple :

```
# stack_trace.py
from corelibs import log, config

# config.DEFAULT_STACK_TRACE = True  # on force globalement l'affichage détaillé␣
↪des piles d'exécution


def palindrome(mot):
    mots = list(mot)
    len_mot = len(mots)
    for i in range(int(len_mot / 2) + 1):
        if mot[i].lower() != mot[(len_mot - 1) - i].lower():
            return False

    return True


@log.stack_trace(force=True)  # on force localement l'affichage détaillé des␣
↪piles d'exécution
def is_palindrome(mot):
    if palindrome(mot):
        print("\"{mot}\" est un palindrome".format(mot=mot))
    else:
        print("\"{mot}\" n'est pas un palindrome".format(mot=mot))


is_palindrome("Hello Kim, c'est papa =}")  # False
is_palindrome("saippuakauppias")  # True
```


Run:
stack_trace
args = ('saippuakauppias',)
kwargs = {}
config.DEFAULT_STACK_TRACE = False
force = True
tp = <stackprinter.tracing.TracePrinter object at 0x0000023E1EEB7100>
tb.TracePrinter = <class 'stackprinter.tracing.TracePrinter'>
tp.enable = <method 'TracePrinter.enable' of <stackprinter.tracing.TracePrinter object at 0x0000023E1EEB7100> tracing.py:105>
wrapped_func = <function 'is_palindrome' stack_trace.py:19>
tp.disable = <method 'TracePrinter.disable' of <stackprinter.tracing.TracePrinter object at 0x0000023E1EEB7100> tracing.py:115>
File "D:/OneDrive/Documents/[PYTHON_PROJECTS]/corelibs/tests/log\stack_trace.py", line 19, in is_palindrome
--> 19 @log.stack_trace(force=True) # on force localement l'affichage détaillé des piles d'exécution
20 def is_palindrome(mot):
log.stack_trace = <function 'stack_trace' log.py:866>
mot = 'saippuakauppias'
File "D:/OneDrive/Documents/[PYTHON_PROJECTS]/corelibs/tests/log\stack_trace.py", line 21, in is_palindrome
19 @log.stack_trace(force=True) # on force localement l'affichage détaillé des piles d'exécution
20 def is_palindrome(mot):
--> 21 if palindrome(mot):
22 print("\"{mot}\" est un palindrome".format(mot=mot))
log.stack_trace = <function 'stack_trace' log.py:866>
mot = 'saippuakauppias'
File "D:/OneDrive/Documents/[PYTHON_PROJECTS]/corelibs/tests/log\stack_trace.py", line 7, in palindrome
--> 7 def palindrome(mot):
8 mots = list(mot)
mot = 'saippuakauppias'
File "D:/OneDrive/Documents/[PYTHON_PROJECTS]/corelibs/tests/log\stack_trace.py", line 16, in palindrome
7 def palindrome(mot):
(...)
12 continue
13 else:
14 return False
15
--> 16 return True
mot = 'saippuakauppias'
Return True
"saippuakauppias" est un palindrome

corelibs.log.**status_bar**(*status_bar_title=None*)

Description

Décorateur pour figer sur le terminal, tout en bas, l'état d'avancement.

La barre de statut n'existe que pendant le temps de l'exécution de la fonction décorée (i.e. lors de l'appel d'une autre fonction, cette barre disparait si cette dernière fonction n'est elle-même pas décorée).

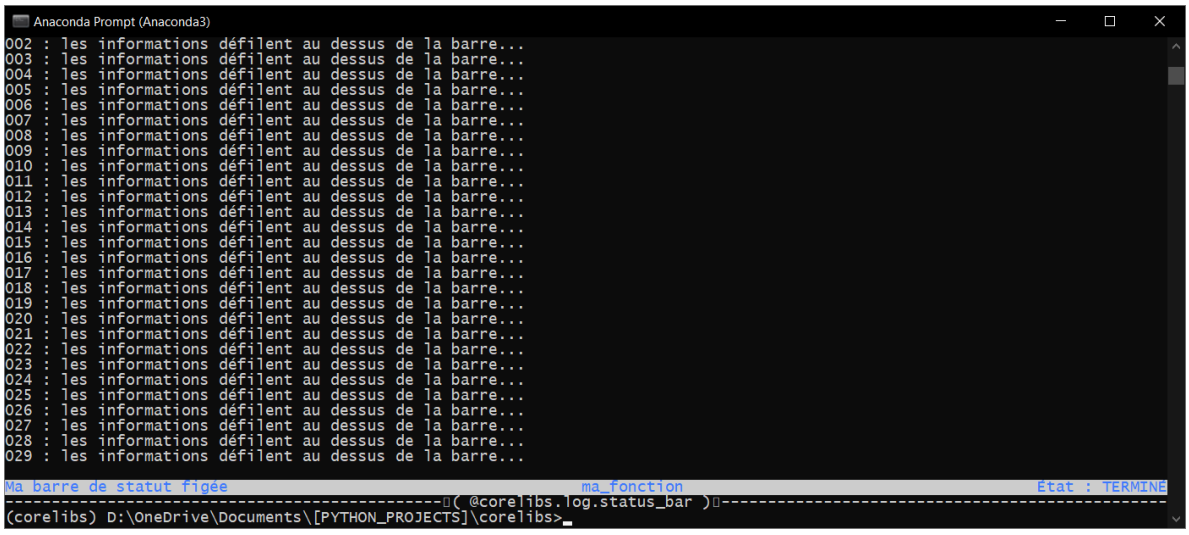
Parameters **status_bar_title** – permet de donner un label à la barre de statut

Returns

rien...

Exemple :

```
# status_bar.py
from corelibs import log


# Ajout décorateur status_bar() pour afficher des informations dans le terminal
# /!\ ATTENTION /!\
#     Jupyter Notebook n'étant pas un terminal, il ne se passera rien... =}
@log.status_bar("Ma barre de statut figée")
def ma_fonction_avec_barre_statut(nb):
    for _ in range(nb):
        print("{_} : les informations défilent au dessus de la barre...".format(_=
↪"{:0>3}".format(_)))


ma_fonction_avec_barre_statut(73)
```



corelibs.log.**timing**(*log_handler=None*)

Description

Décorateur pour chronométrer le temps d'exécution de toutes fonctions cibles

---

Parameters **log_handler** – indique si le timing doit faire une sortie dans un fichier log, défini via l'instanciation de cf. *ColorLog* ou cf. *TermColorLog*.

Returns

rien...

Exemple :

```
# timing.py
from corelibs import log
from numba import njit


cl = log.ColorLog()  # 1. créer une instance de ColorLog()


# Ajout décorateur timing() pour calculer le temps d'exécution de la fonction `ma_
↪premiere_fonction_optimisee()`
@log.timing()
@njit()
def ma_premiere_fonction_optimisee(nb):
    total = 0
    for _ in range(nb):
        total = total + 1

    return total


cl.info(
    "le total est "
    + "{0:,}".format(
        ma_premiere_fonction_optimisee(30000000000)
    ).replace(",", " ")
)


# Faire sortir le résultat du décorateur timing() dans un fichier log spécifique,␣
↪ici, l'objet `cl`
@log.timing(cl)
def ma_deuxieme_fonction_non_optimisee(nb):  # définition d'une 2ème fonction...
    total = 0
    for _ in range(nb):
        total = total + 1
```

(continues on next page)

(continued from previous page)

```
    return total


cl.info(
    "le total est "
    + "{0:,}".format(
        ma_deuxieme_fonction_non_optimisee(30000000000)
    ).replace(",", " ")
)
```

CHAPTER

FIVE

MODULE TOOLS

Description générale

Module pour gérer les compressions, diagnostics, calcul des tailles occupées etc…

corelibs.tools.**get_file_properties**(*file_path*, *pretty_byte_size=True*, *continue_on_error=False*)

Description

Récupère les informations d'un fichier ou d'un répertoire

Parameters

- **file_path** – chemin absolu du fichier/répertoire
- **pretty_byte_size** – afficher la taille du fichier/répertoire de manière lisible pour un humain

  valeurs possibles: *False/True*
  valeur par défaut: *True*

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

tuple nommé avec comme attributs :

- st_mode (mode de protection en binaire)
- st_ino (n° inode)

- st_dev (n° machine)
- st_nlink (nb de liens)
- st_uid (propriétaire)
- st_gid (groupe id sous Windows ou groupe sous Unix)
- st_size (taille du fichier/répertoire)
- st_atime (date dernier accès)
- st_mtime (date dernière modification)
- st_ctime (date de création sous Windows et date dernier modif/accès sous Unix)

Exemple :

```
# get_file_properties.py
from corelibs import tools as to


# récupération des informations par défaut (affichange la taille dynamiquement, à␣
↪la valeur la plus appropriée)
filename = r"D:\OneDrive\Documents\_TEST_\2020-11-11.jpg"
filename_properties = to.get_file_properties(filename)
print(filename_properties)
# affiche l'objet FileProperties(
#   st_mode=33206,
#   st_ino=11540474045256220,
#   st_dev=3199390331,
#   st_nlink=1,
#   st_uid='Invités',
#   st_gid=0,
#   st_size='96.26 Ko',  # la taille est exprimée en Ko pour une petite image
#   st_atime='13/11/2020 22:22:21',
#   st_mtime='11/11/2020 22:10:46',
#   st_ctime='11/11/2020 22:10:39')

# récupération des informations par défaut (affichange la taille dynamiquement, à␣
↪la valeur la plus appropriée)
filename = r"D:\OneDrive\Documents\_TEST_\Zatoichi.avi"
filename_properties = to.get_file_properties(filename)
print(filename_properties)
# affiche l'objet FileProperties(
#   st_mode=33206,
#   st_ino=1125899906961926,
#   st_dev=3199390331,
#   st_nlink=1, st_uid='miche',
#   st_gid=0,
#   st_size='699.78 Mo',  # la taille est exprimée en Mo pour le film Zatoichi.avi
#   st_atime='13/11/2020 22:19:27',
#   st_mtime='22/03/2013 20:17:02',
#   st_ctime='10/11/2020 15:56:52')

# récupération des informations avec la taille ventilée par unité de mesure (jusqu
↪'au To)
# fichier plus gros
filename = r"D:\OneDrive\Documents\_TEST_\Zatoichi.avi"
filename_properties = to.get_file_properties(filename, pretty_byte_size=False)
```

(continues on next page)

(continued from previous page)

```
print(filename_properties)
# affiche l'objet FileProperties(
#   st_mode=33206,
#   st_ino=1125899906961926,
#   st_dev=3199390331,
#   st_nlink=1,
#   st_uid='miche',
#   st_gid=0,
#   st_size=ByteSize(  # objet ByteSize contenant toutes les tailles converties␣
↪depuis le nb d'octets
#       byte=733775872,
#       kilobyte=716578.0,
#       megabyte=699.78,
#       gigabyte=0.68,
#       terabyte=0.0),
#   st_atime='13/11/2020 22:19:27',
#   st_mtime='22/03/2013 20:17:02',
#   st_ctime='10/11/2020 15:56:52')
# pour accéder à la taille en gigaoctet (gigabyte) par exemple récupérer␣
↪directement son attribut
print(
    "La faille en Go du fichier Zatoichi.avi est de {taille} Go"
    .format(taille=filename_properties.st_size.gigabyte)
)  # affichera "La faille en Go du fichier Zatoichi.avi est de 0.68 Go"

# récupération des informations par défaut (affichange la taille dynamiquement, à␣
↪la valeur la plus appropriée)
filename = r"D:\OneDrive\Documents\_TEST_"
filename_properties = to.get_file_properties(filename)
print(filename_properties)
# affiche l'objet FileProperties(
#   st_mode=16895,
#   st_ino=281474976804077,
#   st_dev=3199390331,
#   st_nlink=1,
#   st_uid='miche',
#   st_gid=0,
#   st_size='4.12 Go',  # la taille est exprimée en Go pour le répertoire _TEST_
#   st_atime='14/11/2020 18:23:11',
#   st_mtime='14/11/2020 17:58:08',
#   st_ctime='07/10/2020 22:17:12')
```

corelibs.tools.**get_fingerprint**(*obj*, *algorithm='sha256'*, *eval_as_string=False*, *continue_on_error=False*)

Description

Calculer l'empreinte digitale (signature numérique) d'une chaîne de caractères ou d'un fichier passé en argument

---

Parameters

- **obj** –

  chaîne de caractères
  ou
  chemin absolu du fichier

- **algorithm** – applique l'algorithme de hashage

  valeurs possibles: *'blake2b', 'blake2s', 'md5', 'sha1', 'sha224', 'sha256', 'sha384', 'sha512', 'sha3_256', 'sha3_224', 'sha3_384' ou 'sha3_512'*
  valeur par défaut: *sha256*

- **eval_as_string** – permet de forcer l'évaluation comme étant une chaine de caractère (si l'objet passé en argument a un nom contenant des caractères \ ou / et n'est pas en réalité un chemin de fichiers)

  valeurs possibles: *False/True*
  valeur par défaut: *False*

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns empreinte digital unique

Exemple :

```
# get_fingerprint.py
from corelibs import tools as to


# Même empreinte = Même fichier (quelque soit son nom, sa date de modif,␣
↪propriétaire etc...)
# fichier 1
filename = r"D:\OneDrive\Documents\_TEST_\2020-11-11.jpg"
print(to.get_file_properties(filename))  # FileProperties(st_mode=33206, st_
↪ino=11540474045256220, st_dev=3199390331, st_nlink=1, st_uid='Invités', st_
↪gid=0, st_size='96.26 Ko', st_atime='14/11/2020 22:12:45', st_mtime='11/11/2020␣
↪22:10:46', st_ctime='11/11/2020 22:10:39')
print(to.get_fingerprint(filename))  #␣
↪b6bbd28aebe109adda9bc16a8f838a5043d7980968a37a5d48f890f7fb4d89dc

# fichier 1 copié et les propriétés affichent bien une différence
copy_filename = r"D:\OneDrive\Documents\_TEST_\2020-11-11 - Copie.jpg"
print(to.get_file_properties(copy_filename))  # FileProperties(st_mode=33206, st_
↪ino=19984723346576074, st_dev=3199390331, st_nlink=1, st_uid='miche', st_gid=0,␣
↪st_size='96.26 Ko', st_atime='14/11/2020 22:06:59', st_mtime='11/11/2020␣
↪22:10:46', st_ctime='14/11/2020 21:56:29')
```

(continues on next page)

(continued from previous page)

```
print(to.get_fingerprint(copy_filename))  #␣
↪b6bbd28aebe109adda9bc16a8f838a5043d7980968a37a5d48f890f7fb4d89dc


# algorithme disponible pour le hashage
print(to.get_fingerprint(filename, algorithm="blake2b"))  #␣
↪0a0ba79c2e6535b256c110bf823ea21b32fa7d1fdbeb16374ec3437269ad1f01e4dfb6a72b0dab929807737a331ac6
print(to.get_fingerprint(filename, algorithm="blake2s"))  #␣
↪0c6b739b821f0fb560e5545445e3a4c600ffc171e506ff307be0528f9d274b56
print(to.get_fingerprint(filename, algorithm="md5"))  #␣
↪62383435cc0519bad598cc51783c5d47
print(to.get_fingerprint(filename, algorithm="sha1"))  #␣
↪f50a38a0c0952f96ae98aa54da7929bfb463f8dd
print(to.get_fingerprint(filename, algorithm="sha224"))  #␣
↪c176d2d98a4f8563762ce3863efb942dbb9c496b4bee89a0374e42fa
print(to.get_fingerprint(filename, algorithm="sha256"))  #␣
↪b6bbd28aebe109adda9bc16a8f838a5043d7980968a37a5d48f890f7fb4d89dc
print(to.get_fingerprint(filename, algorithm="sha384"))  #␣
↪c839a0c58ef1ba3a7577de093f3d0e3746600b7b28fa08881f62aa7d4871220b9373ebdc620704e84c965d972cf887
print(to.get_fingerprint(filename, algorithm="sha512"))  #␣
↪39e72a89e167aa86f998e61763b0b43619cb6347118dd5f6713f830688b7e16cc8b006e8027a3bf1b0c260d24a76a1
print(to.get_fingerprint(filename, algorithm="sha3_256"))  #␣
↪e0009df152ee03665a3f26bc434d143873f2abc900e85e4e45d85608d6fd1787
print(to.get_fingerprint(filename, algorithm="sha3_224"))  #␣
↪0f6ab25b4406ed8f98decae73375451286166d2e3d01095c15530069
print(to.get_fingerprint(filename, algorithm="sha3_384"))  #␣
↪3d3d00f27d1663273d80cd9d4f1f55cf2f185d12f7c4387a77e91daae5ff179ecf8692c975b2f86b92010de5202ce3
print(to.get_fingerprint(filename, algorithm="sha3_512"))  #␣
↪ec208132962c717b6f92105ff04a2af93e2195fca22babc17b2b5ddb784b05c913b86cf09dd37e4ec699e342056051

# Empreinte d'une chaine de caractère
str_2_hash = "Hello Kim!"
print(
    "L'empreinte SH256 de \"{str_2_hash}\" est \"{str_hashed}\""
    .format(
        str_2_hash=str_2_hash,
        str_hashed=to.get_fingerprint(str_2_hash)
    )
)  # L'empreinte SH256 de "Hello Kim!" est
↪"8c9affc6a8329ea9c8a87cfe989565c21b24136dc57ccf9407aeefbcac87f97a"

str_2_hash = "/!\\ ATTENTION /!\\"  # doit être forcé comme chaine de caractères␣
↪pour l'évaluation car contient des caractères \ et /
print(
    "L'empreinte SH256 de \"{str_2_hash}\" est \"{str_hashed}\""
    .format(
        str_2_hash=str_2_hash,
        str_hashed=to.get_fingerprint(str_2_hash, eval_as_string=True)
    )
)  # L'empreinte SH256 de "/!\ ATTENTION /!\" est
↪"20f2328fcc5d513e330f5badadac2c0bd2685d8a21ce49daf7690cc6e783ff65"
```

corelibs.tools.**get_total_lines_in_file**(*file*, *size=65536*)

Description

Permet de compter le nombre de lignes dans un fichier plat

Parameters

- **file** – indique l'emplacement du fichier avec son chemin absolu
- **size** – indique le buffer de lecture.

  valeur par défaut: *65536*

Returns

total lignes lues

Exemple :

```
# get_total_lines_in_file.py
from corelibs import tools as to

print(
    "Total lignes lues :",
    to.get_total_lines_in_file(r"\\wsl$\Ubuntu-20.04\root\.zsh_history")
)  # affiche Total lignes lues : 46

print(
    "Total lignes lues :",
    to.get_total_lines_in_file(r"D:\OneDrive\Documents\_TEST_\SAS\Librairie_SAS\
↪00_LIB_Macros_Communes.sas")
)  # affiche Total lignes lues : 4060
```

corelibs.tools.**get_total_lines_in_folder**(*dir_2_scan*, *files_pattern*, *to_exclude=None*)

Description

Permet de compter le total de nombres de lignes de tous les fichiers plats dans un dossier, avec possibilités de filtrage

---

Parameters

- **dir_2_scan** – indique l'emplacement du répertoire à scanner en chemin absolu
- **files_pattern** – indique le schéma du scan.
- **to_exclude** – indique les exclusions, qu'elles soient des sous dossiers et/ou des fichiers. Les exclusions peuvent être des schémas d'exclusions.

Returns

tuple nommé avec comme attributs :

- total_files (total de fichiers lus)
- total_lines (total de lignes lues)

Exemple :

```
# get_total_lines_in_folder.py
from corelibs import tools as to

# Scan standard...
wcl = to.get_total_lines_in_folder(r"\\wsl$\Ubuntu-20.04\root", "*history")
print(wcl)  # affiche un objet Wcl(total_files=3, total_lines=431)

################################################################################
↪#####################################

# Scan avec multiple schéma et exclusion répertoires et/ou fichiers
# étalon *.sas
wcl = to.get_total_lines_in_folder(
    dir_2_scan=r"D:\OneDrive\Documents\_TEST_\SAS_ADD",
    files_pattern="*.sas"
)
print(wcl)  # affiche Wcl(total_files=95, total_lines=20943)
#
# étalon *.log
wcl = to.get_total_lines_in_folder(
    dir_2_scan=r"D:\OneDrive\Documents\_TEST_\SAS_ADD",
    files_pattern="*.log"
)
print(wcl)  # affiche Wcl(total_files=3, total_lines=587)
#
# scan multiple
wcl = to.get_total_lines_in_folder(
    dir_2_scan=r"D:\OneDrive\Documents\_TEST_\SAS_ADD",
    files_pattern=("*.sas", "*.log")
)
print(wcl)  # affiche Wcl(total_files=98, total_lines=21530) qui est bien le␣
↪total des 2 sous ensembles étalons =)

################################################################################
↪#####################################

# Scan multiple avec exclusions
# étalon 1
wcl = to.get_total_lines_in_folder(
    dir_2_scan=r"D:\OneDrive\Documents\_TEST_\SAS_ADD\_QUAL_",
    files_pattern=("*.sas", "*.log")
)
print(wcl)  # Wcl(total_files=93, total_lines=20869)

# étalon 2
wcl = to.get_total_lines_in_folder(
    dir_2_scan=r"D:\OneDrive\Documents\_TEST_\SAS_ADD\[ _BACKUP_POINT_0 ]",
    files_pattern=("*.sas", "*.log")
)
print(wcl)  # Wcl(total_files=4, total_lines=654)
```

(continues on next page)

(continued from previous page)

```
# étalon 3
print(
    "Total lignes lues :",
    to.get_total_lines_in_file(r"D:\OneDrive\Documents\_TEST_\SAS_ADD\hello.sas")
)  # Total lignes lues : 7

# étalon 4
wcl = to.get_total_lines_in_folder(
    dir_2_scan=r"D:\OneDrive\Documents\_TEST_\SAS_ADD",
    files_pattern=("*.sas", "*.log")
)
print(wcl)  # Wcl(total_files=98, total_lines=21530)
# 98 = 93 + 4 + 1
# 21530 = 20869 + 654 + 7

# exclusions étalon
wcl = to.get_total_lines_in_folder(
    dir_2_scan=r"D:\OneDrive\Documents\_TEST_\SAS_ADD",
    files_pattern=("*.sas", "*.log"),
    to_exclude=r"*_BACKUP_POINT_0*"
)
print(wcl)  # Wcl(total_files=94, total_lines=20876)
# 94 = 98 - 4
# 20876 = 21530 - 654

# exclusions finales
wcl = to.get_total_lines_in_folder(
    dir_2_scan=r"D:\OneDrive\Documents\_TEST_\SAS_ADD",
    files_pattern=("*.sas", "*.log"),
    to_exclude=(r"*_BACKUP_POINT_0*", r"*SAS_ADD\hello.sas")
)
print(wcl)  # Wcl(total_files=93, total_lines=20869)
# 93 = 98 - 4 - 1
# 20869 = 21530 - 654 - 7

# le compte y est! =}
```

class corelibs.tools.**Archive**

unzip(*yaml_file=None*, *archive_name=None*, *files_2_unzip=None*, *continue_on_error=False*)

Description

Permet de décompresser les données

Parameters

- **yaml_file** – fichier de configuration pour décompresser ; si utilisé en argument, les 2 arguments archive_name et files_2_unzip ne seront pas pris en compte. (ce fichier peut être créé manuellement ou être celui généré par la méthode *corelibs.tools.Archive.zip()*)

- **archive_name** – nom du fichier archive (avec son ou ses extensions) contenant les fichiers à décompresser **avec** son chemin absolu (i.e. "D:\OneDrive\Documents_TEST_\NOM_ARCHIVE.7z")
- **files_2_unzip** – le ou les fichiers à décompresser

  si fichier simple, alors définir avec une chaîne de caractères
  si liste de fichiers alors définir les différentes chaînes de caractères dans un tuple, séparé par des virgules (i.e. ("fichier 1", "fichier 2", . . . , "fichier n"))
  les noms des fichiers peuvent être précédés par un chemin ou non :

  - sans le chemin en préfixe, alors l'emplacement du fichier à décompresser est celui du fichier archive
  - sinon le chemin en préfixe sera pris en priorité sur le chemin de l'archive pour la décompression
- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

  valeurs possibles: *False/True*
  valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

code retour :

- 0 si OK
- <> 0 si KO

Exemple :

```
# unarchive.py
from corelibs import tools as to


archive = to.Archive()
# Décompression via un fichier de configuration YAML
exit_code = archive.unzip(yaml_file=r"D:\OneDrive\Documents\_TEST_\TEST.yaml")

# Décompression manuelle
exit_code = archive.unzip(
    archive_name=r"D:\OneDrive\Documents\_TEST_\\/TEST.28112013.7z",  # pas␣
↪beau, mais on peut rentrer ce que l'on veut
    files_2_unzip=(
        #  fichiers décompressés à la racine de l'archive (i.e. "D:\OneDrive\
↪Documents\_TEST_\\/")
        r"D:\OneDrive\Desktop\1-01 Act One Questo Mar Rosso.m4a",
        "_éèçàoöôîïêëùûü;.txt",
        # fichier décompressé à un emplacement spécifique, ici "D:\OneDrive\
↪Documents\_TEST_\A zip"
```

(continues on next page)

(continued from previous page)

```
        r"D:\OneDrive\Documents\_TEST_\dossier compressé"
    )
)
```

zip(*yaml_file=None*, *archive_name=None*, *files_2_zip=None*, *delete_sources_files=True*, *continue_on_error=False*)

Description

Permet de compresser les données

Parameters

- **yaml_file** – fichier de configuration pour compresser ; si utilisé en argument, les 2 arguments archive_name et files_2_zip ne seront pas pris en compte.

  Un fichier de configuration ayant pour nom "archive_name.YAML" sera généré à la racine de archive_name et peut être utilisé en argument pour :

  - compresser en mode mise à jour
  - décompresser (cf. *corelibs.tools.Archive.unzip()*)

- **archive_name** – nom souhaité pour l'archive **avec** son chemin absolu pour le stockage (i.e. "D:\OneDrive\Documents_TEST_\NOM_ARCHIVE")
- **files_2_zip** – le ou les fichiers à compresser

  si fichier simple, alors définir avec une chaîne de caractères
  si liste de fichiers alors définir les différentes chaînes de caractères dans un tuple, séparé par des virgules (i.e. ("fichier 1", "fichier 2", . . . , "fichier n"))
  les noms des fichiers peuvent être précédés par un chemin ou non :

  - sans le chemin en préfixe, alors l'emplacement du fichier à ziper est celui du fichier archive
  - sinon le chemin en préfixe sera pris en priorité sur le chemin de l'archive

- **delete_sources_files** – supprimer les fichiers sources une fois la compression finie (i.e. terminée sans erreur ou sans avoir été arrêtée par un autre processus)

  valeurs possibles: *False/True*
  valeur par défaut: *True*

- **continue_on_error** – forcer l'exécution lorsqu'une erreur est levée

valeurs possibles: *False/True*
valeur par défaut: *False* (cf. *DEFAULT_CONTINUE_ON_ERROR* dans *Configurations*)

Returns

code retour :

- 0 si OK
- <> 0 si KO

Exemple :

```
# archive.py
from corelibs import lazy as lz, tools as to


archive = to.Archive()
# Compression via un fichier de configuration YAML
exit_code = archive.zip(yaml_file=r"D:/OneDrive//\\//Documents\_TEST_\TEST.
↪yaml")  # pas beau, mais on peut rentrer ce que l'on veut...

# Compression manuelle
exit_code = archive.zip(
    archive_name=r"D:/OneDrive/Documents\_TEST_\MON_ARCHIVE_" + lz.get_
↪timestamp(),
    files_2_zip=(
        #  fichiers présents à la racine de l'archive (i.e. "D:/OneDrive/
↪Documents\_TEST_")
        "1-01 Act One Questo Mar Rosso.m4a",
        "_éèçàoöôîïêëùûü;.txt",
        # fichier à un emplacement spécifique, ici "D:\OneDrive\Documents\_
↪TEST_\A zip"
        r"D:\OneDrive\Documents\_TEST_\A zip\A zip - Copie"
    )
)
```

CHAPTER

SIX

DÉPENDANCES

Description générale

Les dépendances sont installées automatiquement lors de l'installation/MAJ du package corelibs.

Dans le cas d'une création manuelle d'un nouvel environnement virtuel, selon le contexte de création, il est peut-être nécessaire de réinstaller manuellement les dépendances ci-dessous dans le-dit nouvel environnement.

Note: Pour mémoire, l'installation d'un package se fait avec cette commande

```
$ pip install nom_package
```

Et la mise à jour d'un package se fait avec cette commande

```
$ pip install nom_package -U
```

CHAPTER

SEVEN

LIENS UTILES

Description générale

Tous les liens utiles sont listés ici, à jeter un oeil (voire même les 2 =þ)

- **Python 3.* :**
  - –
  - –
  - –
- **Normes/Spec/Documentations :**
  - –
  - –
  - –
  - –
  - –

CHAPTER

EIGHT

INDEX & TABLES DES MATIÈRES

- genindex
- modindex
- search

PYTHON MODULE INDEX

c

l

t

INDEX