exo3-solution

October 9, 2023

1 Exercice 1 : librairie time

[13]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Code qui permet de fournir le jour de la semaine à partir d'une date donnée au␣
 ↪format dd-mm-yyyy (mois-jour-année).
"""

import time


def find_weekday(str_date):
    """
    à partir d'une date `str_date` (str) au format dd-mm-yyyy,
    retourne
    - une chaîne de caractères avec le jour de la semaine correspondant
    """
    time_object = time.strptime(str_date, "%d-%m-%Y")
    day_indix = time.strftime("%w", time_object)
    weekdays = ["Dimanche", "Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi",␣
 ↪"Samedi"]
    return weekdays[int(day_indix)]


if __name__ == "__main__":
    day = "04-07-2012"
    print("Le {} était un {}".format(day, find_weekday(day)))
```

```
Le 04-07-2012 était un Mercredi
```

2 Exercice 2 : librairie os

[28]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

```
Code qui permet de créer des fichiers vides au format  'output-dd-mm-yyyy-XX.
 ↪txt' dans un répertoire `test`
"""

import time
import os


def make_files(N=25, str_date=time.strftime("%d-%m-%Y")):
    """
    Fonction qui créé N fichiers de la forme 'output-dd-mm-yyyy-XX.txt'
    dans le dossier `test` situé dans le répertoire du script d'exécution␣
 ↪(sinon, le dossier est créé)
        où dd-mm-yyyy est la date au format jour-mois-année indiquée par␣
 ↪str_date.

    input parameters :
    - N (int) : nombre de fichiers à créer
    - str_date (str) : date à mettre dans les noms de fichier, date du jour par␣
 ↪défaut.
    """
    # Création du répertoire test s'il n'existe pas déjà
    if not os.path.exists("./test"):
        os.mkdir("test")

    filenames = []
    for i in range(N):
        # attention à bien ajouter 1 pour avoir un premier numéro égal à 1 et␣
 ↪s'arrêter à N
        filename = "test/output-{}-{}.txt".format(str_date, i + 1)
        filenames.append(filenames)
        # On créé un fichier vide
        with open(filename, "w") as f:
            pass

    return filenames


if __name__ == "__main__":
    make_files()
    make_files(str_date="21-03-2023")
    make_files(str_date="10-05-2024")
    make_files(str_date="30-04-2022")
```

Il faut changer le format des fichiers en : 'output-yyyy-mm-dd-XXXX.txt' où XXXX est maintenant une chaîne de caractères avec des zéros qui servent pour faire du padding.

[29]:
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Code qui permet de créer des fichiers vides au format  'output-dd-mm-yyyy-XX.txt' dans un répertoire `test`
"""

import time
import os
import glob


def rename_files(str_date=time.strftime("%d-%m-%Y")):
    """
    Fonction qui renomme N fichiers de la forme 'output-dd-mm-yyyy-XX.txt'
    dans le dossier `test` situé dans le répertoire du script d'exécution (sinon, le dossier est créé)
    au format  'output-yyyy-mm-dd-XXXX.txt' où XXXX est le

    input parameters :
    - str_date (str) : date correspondant aux fichiers à renommer, date du jour par défaut.
    """

    filenames = glob.glob("test/output-{}-*.txt".format(str_date))
    for old_filename in filenames:
        time_obj = time.strptime(str_date, "%d-%m-%Y")
        str_date_new = time.strftime("%Y-%m-%d", time_obj)
        indix = int(old_filename.split("-")[-1].replace(".txt", ""))
        new_filename = "test/output-{}-{:04d}.txt".format(str_date_new, indix)
        os.rename(old_filename, new_filename)
    return filenames


if __name__ == "__main__":
    if os.path.exists("./test"):
        rename_files()
        rename_files(str_date="21-03-2023")
        rename_files(str_date="10-05-2024")
        rename_files(str_date="30-04-2022")
    else:
        print("Le répertoire `test` n'existe pas")
```

2.1 Exercice 3

[7]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
calcule la position d'équilibre du cerceau en fonction des paramètres d'entrées␣
 ↪en unité MKSA

Les paramètres g, r, w peuvent être changés lors de l'appel du script
"""

# Importation des librairies
import math
import os
import argparse

# Definition des fonctions
def posEquilibre(g, r, w):
    """
    calcule la position d'équilibre du cerceau en fonction des paramètres␣
 ↪d'entrées en unité MKSA
    - g (float) : gravité en m^2/s
    - r (float) : rayon du cerceau (m)
    - w (float) : vitesse de rotation en rad/s
    retourne
    - thetaEq (float) en radian
    - w0 (float) en rad/s
    """
    wc = math.sqrt(g / r)
    if w <= wc:
        thetaEq = math.pi
        w0 = wc * math.sqrt(1.0 - (w / wc) ** 2)
    else:
        thetaEq = math.acos(-((wc / w) ** 2))
        w0 = w * math.sqrt(1.0 - (wc / w) ** 4)
    return thetaEq, w0


def readingParameters(filename):
    """
    format attendu : parametername value
    avec parametername qui vaut gravity, radius ou omega et value un nombre réel
    """
    r = 1.0
    g = 9.81
    w = 2.0
```

```
    argFound = 0
    if os.path.exists(filename):
        with open(filename, "r") as file:
            for line in file:
                if "gravity" in line:
                    g = float(line.split()[1])
                    argFound += 1
                if "radius" in line:
                    r = float(line.split()[1])
                    argFound += 1
                if "omega" in line:
                    w = float(line.split()[1])
                    argFound += 1
    else:
        print(
            "Le fichier d'entrée n'existe pas, les valeurs par défaut seront␣
 ↪utilisées"
        )

    if argFound == 3:
        return g, r, w
    else:
        print(
            "Le fichier d'entrée contient moins de paramètres que le minimum␣
 ↪nécessaire, les valeurs par défaut seront utilisées pour les valeurs␣
 ↪manquantes"
        )
        return g, r, w


# Programme principal
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-g", "--gravity", default=9.81, type=float, help="gravité en m^2/s"
    )
    parser.add_argument("-r", "--radius", default=1.0, type=float, help="rayon␣
 ↪en m")
    parser.add_argument(
        "-w", "--omega", default=2.0, type=float, help="pulsation en rad/s"
    )
    parser.add_argument("-f", "--inputfile", type=str, help="fichier d'input")
    args = parser.parse_args()

    if args.inputfile is not None:
        print(
```

```
            "Tous les arguments sont lus à partir du fichier {}".format(args.
↪inputfile)
        )
        g, r, w = readingParameters(args.inputfile)
    else:
        g, r, w = args.gravity, args.radius, args.omega
    thetaEq, w0 = posEquilibre(g, r, w)
    print(
        "Les paramètres du probleme sont : g = {} r = {} w = {} [MKSA]".
↪format(g, r, w)
    )
    print(
        "La position d'équilibre stable est : theta_eq = {} \n\
    La pulsation des petites oscillations autour de cet équilbre est : omega_0␣
↪= {} [MKSA]".format(
            thetaEq, w0
        )
    )
```

```
Tous les arguments sont lus à partir du fichier /home/mverot/.local/share/jupyte
r/runtime/kernel-d07309f8-8ab2-4224-98e6-8c8436b62cb7.json
Le fichier d'entrée contient moins de paramètres que le minimum nécessaire, les
valeurs par défaut seront utilisées pour les valeurs manquantes
Les paramètres du probleme sont : g = 9.81 r = 1.0 w = 2.0 [MKSA]
La position d'équilibre stable est : theta_eq = 3.141592653589793
La pulsation des petites oscillations autour de cet équilbre est : omega_0 =
2.41039415863879 [MKSA]
```

```
[ ]:
```