Programmes Python

Dylan Robins - E2I4 - 2021-03-24

1. build_md.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import os
from datetime import datetime

def search_dir(output_file, cwd, h_level=2):

    for i, item in enumerate(sorted(os.listdir(cwd))):
        path = os.path.join(cwd, item)
        # if item is a directory, descend into it and list it's contents
        if os.path.isdir(path):
            output_file.write('#' * h_level + f" {i+1}. {item}\n\n")
            search_dir(output_file, path, h_level + 1)

        elif os.path.isfile(path) and os.path.splitext(path)[-1] == ".py":
            output_file.write('#' * h_level + f" {i+1}. {item}\n\n")
            output_file.write("```python\n")
            with open(path, 'r') as py_file:
                output_file.write(py_file.read() + "\n")
            output_file.write("```\n\n")


with open("python_programs.md", "w") as md_file:
```

```
    md_file.write(
        f"# Programmes Python\n"
        f"\n"
        f"Dylan Robins - E2I4 - {datetime.now().date().isoformat()}\n"
        f"\n"
    )
    search_dir(md_file, '.')
```

2. les_bases

1. revisions_1

1. agenda_tel.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from color_print import cprint

def saisir_numero(tel):
    tel["code"]= input("Code région (ex: +33) ?\n>>> ")
    tel["num"] = input("Numéro (ex: 412345678) ?\n>>> ")

def saisir_liste_telephones():
    tels = []
    for _ in range(3):
        tels.append({})
        saisir_numero(tels[-1])
    return tels

def saisir_personne():
    p = {}
    p["nom"] = input("Nom de la personne ?\n>>> ")
    p["nums"] = saisir_liste_telephones()
    p["nationalite"] = input(f"Nationalité de {p['nom']} ?\n>>> ")
    return p

def afficher_personne(personne):
    print(f"{personne['nom']} ({personne['nationalite']}) :")
    for tel in personne['nums']:
        print(f"    ", tel['code'], tel['num'], sep='')

def afficher_nationalite(agenda):
    nat = input("Nationalité à rechercher dans l'agenda ?\n>>> ")
    for personne in agenda:
        if personne["nationalite"] == nat:
            afficher_personne(personne)

def chercher_num(agenda):
    num = {}
    saisir_numero(num)
    for personne in agenda:
        if num in personne["nums"]:
            afficher_personne(personne)


if __name__ == "__main__":
```

```python
    print("Création agenda téléphonique :")
    agenda = []
    N = int(input("Nombre de personnes à rajouter à l'agenda : >>> "))
    for _ in range(N):
        agenda.append(saisir_personne())

    for entry in agenda:
        print(entry)
```

2. anagrammes.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from sys import argv
from color_print import cprint

def fact(n):
    f = n
    while n > 1:
        n -= 1
        f *= n
    return f


def anagramme_pos(ch, pos):
    if pos == 0:
        anagrammes = [
            ch[0] + ch[1] + ch[2],
            ch[0] + ch[2] + ch[1]
        ]
    elif pos == 1:
        anagrammes = [
            ch[1] + ch[0] + ch[2],
            ch[1] + ch[2] + ch[0]
        ]
    else:
        anagrammes = [
            ch[2] + ch[0] + ch[1],
            ch[2] + ch[1] + ch[0]
        ]
    return anagrammes


def est_dans_tableau(tab, ch):
    return ch in tab


def remplir_tab_anagrammes(tab, ch):
    for anagram in [anagramme_pos(ch, i) for i in range(len(ch))]:
        tab.append(anagram)

def test():
    print("Anagram generation : ", end="")
    ch = "abc"
    anagrams = [['abc', 'acb'], ['bac', 'bca'], ['cab', 'cba']]
    tab = []
    remplir_tab_anagrammes(tab, ch)

    if tab == anagrams:
        cprint("PASS", 'blue')
    else:
        cprint("FAIL", 'red')

if __name__ == "__main__":
    if len(argv) == 2 and argv[1] == "test":
        test()
    else:
        ch = ""
        tab = []
        while len(ch) != 3:
            ch = input("Entrez un mot de 3 lettres : >>> ")
        remplir_tab_anagrammes(tab, ch)
        print("Anagrams of", ch, ":", tab)
```

3. color_print.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def cprint(text, color):
    cols = {
        "rst": '\033[0m',
        "cyan": '\033[36m',
        "blue": '\033[34m',
        "red": '\033[91m'
    }
    print(cols[color] + str(text) + cols["rst"])
```

4. copie_matrice_2D.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from color_print import cprint

def transpose_matrix(m1, m2):
    for y, row in enumerate(m1):
        x = 0
        while x < len(row):
            m2[len(row)-1 - x][len(m1)-1 - y] = row[x]
            x += 1

def test():
    print("Testing matrix transposition : ", end="")
    m1 = [
        [1, 2, 3],
        [4, 5, 6]
    ]
    m2 = [[None for x in range(2)] for y in range(3)]
    ref = [
        [6, 3],
        [5, 2],
        [4, 1]
    ]

    transpose_matrix(m1, m2)

    if m2 == ref:
        cprint("PASS", 'blue')
    else:
        cprint("FAIL", 'red')

if __name__ == "__main__":
    test()
```

5. enfants_employes.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from color_print import cprint

def saisir_date():
    date = {}
    date["jour"] = int(input("Jour ? >>> "))
    date["mois"] = int(input("Mois ? >>> "))
    date["annee"] = int(input("Année ? >>> "))
    return date

def saisir_enfant():
    enfant = {}
    enfant["prenom"] = input("Prénom de l'enfant ? >>> ")
    enfant["anniv"] = saisir_date()
    return enfant

def saisir_employe():
    employe = {}
    employe["nom"] = input("Nom de l'employé ? >>> ")
    employe["prenom"] = input("Prénom de l'employé ? >>> ")
    employe["nb_enfants"] = int(input("Nombre d'enfants ? >>> "))
    employe["enfants"] = []
    for child in range(employe["nb_enfants"]):
        employe["enfants"].append(saisir_enfant())
    return employe

def saisir_entreprise():
    entreprise = {}
    entreprise["nb_employes"] = int(input("Nombre d'employes ? >>> "))
    entreprise["employes"] = []
    for employe in range(entreprise["nb_employes"]):
        entreprise["employes"].append(saisir_employe())
    return entreprise

def nombre_enfants_entreprise(entreprise):
    return sum(employe["nb_enfants"] for employe in entreprise["employes"])

def compare_dates(d1, d2):
    if d1["annee"] < d2["annee"]:
        return -1
    elif d1["annee"] == d2["annee"] and d1["mois"] < d2["mois"]:
        return -1
```

```python
    elif d1["annee"] == d2["annee"] and d1["mois"] == d2["mois"] and d1["jour"] <
d2["jour"]:
        return -1
    elif d1["annee"] == d2["annee"] and d1["mois"] == d2["mois"] and d1["jour"]
== d2["jour"]:
        return 0
    else:
        return 1

def plus_jeune_enfant(entreprise):
    # Initialiser le plus jeune au 1e enfant du 1e employé
    plus_jeune = entreprise["employes"][0]["enfants"][0]

    # comparer un par un les enfants de tous les employés
    for employe in entreprise["employes"]:
        for child in employe["enfants"]:
            if compare_dates(plus_jeune["anniv"], child["anniv"]) < 0:
                plus_jeune = child
    return plus_jeune

def fdate(date):
    return f"{date['jour']}/{date['mois']}/{date['annee']}"


if __name__ == "__main__":
    entreprise = saisir_entreprise()
    print("")
    print("Enfants :")
    for employe in entreprise["employes"]:
        for child in employe["enfants"]:
            print(f"{child['prenom']} ({fdate(child['anniv'])})")

    child = plus_jeune_enfant(entreprise)
    print(f"Enfant le plus jeune: {child['prenom']}, ({fdate(child['anniv'])})")
```

6. input

7. intouchables.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from math import ceil, sqrt
from sys import argv
from color_print import cprint

def somme_div(y):
    upper = ceil(sqrt(y))+1
    diviseurs = [x for x in range(1, upper) if (y % x == 0)]
    return sum(diviseurs)


def intouchable(x):
    for y in range(2, x ** 2):
        if x == somme_div(y):
            return False
    return True
```

```
def test():
    print("Nombres intouchables : ", end="")

    if not intouchable(4) and not intouchable(16) and intouchable(5) and all(not 
intouchable(x) for x in range(7, 125, 2)):
        cprint("PASS", 'blue')
    else:
        cprint("FAIL", 'red')


if __name__ == "__main__":
    if len(argv) == 2 and argv[1] == "test":
        test()
    else:
        x = input("Number to test (between 0 and 125)?\n>>> ")
        try:
            x = int(x)
            if not (0 < x <= 125):
                raise ValueError

        except ValueError:
            print("Error: couldn't parse", x, "as an integer")

        if intouchable(x):
            print(x, "est intouchable")
        else:
            print(x, "n'est pas intouchable")
```

8. mot_le_plus_long.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def longest_word(ch):
    # split le string into words
    words = ch.split()
    # sort the words by length in descending order
    words.sort(key=len, reverse=True)
    # return the longest word
    return words[0]

if __name__ == "__main__":
    ch = input("Entrez une phrase :\n>>> ")
    print("Mot le plus long:", longest_word(ch))
    print("Longueur du mot le plus long:", len(longest_word(ch)))
```

9. recherche_mots.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from sys import argv
from color_print import cprint

def compter_mots(ch):
    words = ch.split()
    return len(words)

def retourner_mot(ch, pos):
    words = ch.split()
    return words[pos], sum(len(word) for word in words[:pos])

def chercher_mot(ch, nb_tab, mot):
    for i, pair in enumerate(nb_tab):
        if pair["mot"] == mot:
            return i
    return -1

def test():
    print("Recherche de mots : ", end="")
    phrase = "je crois ce que je dis et je fais ce que je crois "
    occurences = [
        {'mot': 'je',    'occurences': 4},
        {'mot': 'crois', 'occurences': 2},
        {'mot': 'ce',    'occurences': 2},
        {'mot': 'que',   'occurences': 2},
        {'mot': 'dis',   'occurences': 1},
        {'mot': 'et',    'occurences': 1},
        {'mot': 'fais',  'occurences': 1}
    ]

    nb_mots = compter_mots(phrase)
    tab = []
    # pour chaque mot
    for i in range(nb_mots):
        # trouver sa 1e occurence dans la chaine
        ieme_mot, pos = retourner_mot(phrase, i)
        # trouver l'indice de ce mot dans notre dict
        idex = chercher_mot(phrase, tab, ieme_mot)
        if idex == -1:
            # 1e occurence du mot: le rajouter à la liste
            tab.append( {"mot": ieme_mot, "occurences": 1} )
        else:
            # incrémenter le compteur
            tab[idex]["occurences"] += 1

    if tab == occurences:
        cprint("PASS", 'blue')
    else:
        cprint("FAIL", 'red')

if __name__ == "__main__":
    if len(argv) == 2 and argv[1] == "test":
        test()
    else:
        phrase = input("Entrez une phrase:\n>>> ")
        nb_mots = compter_mots(phrase)
        tab = []
        # pour chaque mot
        for i in range(nb_mots):
            # trouver sa 1e occurence dans la chaine
            ieme_mot, pos = retourner_mot(phrase, i)
            # trouver l'indice de ce mot dans notre dict
            idex = chercher_mot(phrase, tab, ieme_mot)
            if idex == -1:
                # 1e occurence du mot: le rajouter à la liste
                tab.append( {"mot": ieme_mot, "occurences": 1} )
            else:
                # incrémenter le compteur
                tab[idex]["occurences"] += 1

        for word in tab:
            print(
                f"{word['mot']} : {word['occurences']}",
                "occurence" if word['occurences'] == 1 else "occurences"
            )
```

2. serie1.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
def is_even_and_div_3(n):
    return (n % 2 == 0) and (n % 3 == 0)

def is_between_0_20(n):
    return 0 <= n <= 20

def str_is_div_3(arr):
    n = int(arr)
    return n % 3 == 0

def question4():
    x = input("Enter the value of x : >>> ")
    y = input("Enter the value of y : >>> ")

    print("x =", x, ", y =", y)

    x, y = y, x

    print("x =", x, ", y =", y)

def question5():
    PI = 3.141592653589793
    r = 20
    d = 2 * r
    p = 2 * PI * r
    s = PI * (r ** 2)
    print(f"Un cercle de rayon {r} a pour diamètre {d}, pour circonférence {p} et pour surface {s}")

if __name__ == "__main__":
    print("Test is_even_and_div_3... ", end="")
    assert(is_even_and_div_3(6))
    assert(not is_even_and_div_3(2))
    assert(not is_even_and_div_3(3))
    assert(not is_even_and_div_3(15))
    print("Ok")
    print("Test is_between_0_20... ", end="")
    assert(is_between_0_20(0))
    assert(is_between_0_20(20))
    assert(is_between_0_20(10))
    assert(not is_between_0_20(-1))
    assert(not is_between_0_20(25))
    print("Ok")

    print("Test str_is_div_3... ", end="")
    assert(str_is_div_3("6"))
    assert(str_is_div_3("15"))
    assert(not str_is_div_3("4"))
    print("Ok")

    print("Test question4... ")
    question4()
    print("Test question5... ")
    question5()
```

3. serie2.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
def t_or_f():
    user_data = input("Enter t or f: >>> ")
    if user_data == "t":
        print("True")
    elif user_data == "f":
        print("False")
    else:
        print("Invalid input", user_data)

def quadrant(x, y):
    if x > 0:
        if y > 0:
            print("North east")
        elif y < 0:
            print("South east")
        else:
            print("East")
    elif x < 0:
        if y > 0:
            print("North west")
        elif y < 0:
            print("South west")
        else:
            print("West")
    else:
        if y > 0:
            print("North")
        elif y < 0:
            print("South")
        else:
            print("Center")

def time_in_1s(h, m, s):
    if s < 59:
```

```python
        s += 1
    elif m < 59:
        s = 0
        m += 1
    elif h < 23:
        s = 0
        m = 0
        h += 1
    else:
        s = 0
        m = 0
        h = 0
    print(f"In one second it'll be {h}:{m}:{s}")

def average():
    grades = []
    user_data = 0
    while 0 <= float(user_data) <= 20:
        user_data = input("Enter a grade: >>> ")
        try:
            grades.append(float(user_data))
        except ValueError:
            break
    mean = 0
    for grade in grades:
        mean += grade
    mean = mean / len(grades)
    print("Average grade:", mean)

def disp_range(a, b):
    for i in range(a, b+1):
        print(i, end=", ")
    for i in range(b, a-1, -1):
        print(i, end=", ")
    print("")

if __name__ == "__main__":
    print("Test t_or_f... ")
    t_or_f()
    t_or_f()
    t_or_f()
    t_or_f()

    print("Test quadrant... ")
    quadrant(2, -5)

    print("Test time_in_1s... ")
    time_in_1s(22, 8, 59)

    print("Test average... ")
    average()

    print("Test disp_range... ")
    disp_range(3, 9)
```

4. serie3.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
def quantieme_2020():
    nb_jours = [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

    mois = int(input("Entrez le mois [1-12]: >>> "))
    assert (1 <= mois <= 12), "Mois incorrect"
    jour = int(input(f"Entrez le jour [1-{nb_jours[mois-1]}]: >>> "))
    assert (1 <= jour <= nb_jours[mois-1]), "Jour incorrect"

    quantieme = sum(nb_jours[:mois-1]) + jour
    print(f"Quantième de la date {jour:02d}/{mois:02d}/2020: {quantieme}")

def reverse_table_construction(n):
    tab = []
    for _ in range(n):
        num = int(input("Enter an integer: >>> "))
        tab.insert(0, num)

    print("tab:", tab)
    print("Even elements of tab:", [elem for elem in tab if (elem % 2 == 0)])
    print("Elements at odd indices of tab:", [elem for i,elem in enumerate(tab)
if (i % 2 == 1)])

    x = int(input("Enter an integer to search for: >>> "))
    if x in tab:
        print(x, "is in tab at index", tab.index(x))
    else:
        print(x, "isn't in tab")

    print("Min value in tab:", min(tab))

def create_2d_array(N=2, M=3):
    matrix = [[None for x in range(N)] for y in range(M)]

    y = 0
    while y < M:
        x = 0
        while x < N:
            val = input(f"Enter the value of matrix cell ({x},{y}): >>> ")
            matrix[y][x] = val
            x += 1
        y += 1
    print("Matrix :")
    print_matrix(matrix)

def print_matrix(matrix):
    for row in matrix:
        print("| ", end="")
        for val in row:
            print(val, end=" | ")
        print("")

if __name__ == "__main__":
    print("Test quantieme_2020...")
    quantieme_2020()
    print("Test create_2d_array...")
    create_2d_array()
```

5. serie4.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
entr = {}
entr["nb_employes"] = int(input("Entrez le nombre d'employés : "))

entr["tab_emp"] = [None] * entr["nb_employes"]
for i in range(entr["nb_employes"]):
    print("\nSaisir les informations de l'employe {:d}".format(i+1))
    e = {}
    e["nom"] = input("Entrez son nom : ")
    e["prenom"] = input("Entrez son prénom : ")
    e["nb_enfants"] = int(input("Entrez son nombre d'enfants : "))
    entr["tab_emp"][i] = e

if __name__ == "__main__":
    print("Employés de l'entreprise:")
    total_kids = 0
    for i in range(entr["nb_employes"]):
        employe = entr["tab_emp"][i]
        print(f"L'employé {employe['prenom']} {employe['nom']} a
{employe['nb_enfants']} enfants.")
        total_kids += employe['nb_enfants']

    print(f"Au total, les employés ont {total_kids} enfants.")
```

6. serie5.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##############################################################################
# Question 1
##############################################################################
def produit_val(val):
    val = val * val
    print(type(val))

def question1():
    tab = [2, 3, 6, 8, 10]
    print("Tab initial =", tab)
    produit_val(tab[1])
    print("Tab final =", tab)
    # rien ne change car int immutable

##############################################################################
# Question 2
##############################################################################

def moyenne_3_notes():
    n1 = float(input("Note 1 ? >>> "))
    n2 = float(input("Note 2 ? >>> "))
```

```python
    n3 = float(input("Note 3 ? >>> "))
    return (n1 + n2 + n3) / 3

def moyenne_max_notes():
    nmax = 0
    somme = 0
    for i in range(3):
        note = float(input("Note {} ? >>> ".format(i)))
        if (note > nmax):
            nmax = note
        somme += note
    return somme / 3, nmax

def moyenne_notes():
    somme = 0
    for i in range(3):
        note = float(input("Note {} ? >>> ".format(i)))
        if not (0 < note < 20):
            return -1
        somme += note
    return somme / 3

def question2():
    print("Moyenne: ", moyenne_3_notes())

    moy, nmax = moyenne_max_notes()
    print("Moyenne:", moy, "Maximum:", nmax)

    print("Moyenne: ", moyenne_notes())

################################################################################
# Question 3
################################################################################

def remplir_enseignement(enseignement):
    filiere = input("Nom de la filière? >>> ")
    respo = input("Nom du responsable? >>> ")
    enseignement["filiere"] = filiere
    enseignement["responsable"] = respo

def aff_par_filiere(enseignements):
    for ens in sorted(enseignements, key=lambda ens: ens["filiere"]):
        print(f"{ens['code']}: filiere {ens['filiere']} - reponsable
{ens['responsable']}")

def aff_par_respo(enseignements):
    for ens in sorted(enseignements, key=lambda ens: ens["responsable"]):
        print(f"{ens['code']}: filiere {ens['filiere']} - reponsable
{ens['responsable']}")


def question3():
    enseignements = []
    while len(enseignements) < 50:
        ens = input("Enseignement à remplir ? (Q/q pour quitter) >>> ")
        if ens in ["Q", "q"]:
            break
        enseignements.append( {"code": ens} )
        remplir_enseignement(enseignements[-1])

    print("Affichage par filière :")
    aff_par_filiere(enseignements)

    print("Affichage par responsable :")
    aff_par_respo(enseignements)

if __name__ == "__main__":
    print("Test question 1:")
    question1()
    print("Test question 2:")
    question2()
    print("Test question 3:")
    question3()
```

3. listes_modules_fichiers

1. serie1.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

###############################################################################
# EXO 1
###############################################################################
def exercice1(catalogue):
    matières_théoriques = [ue for ue in catalogue if ue[2] == 'T']
    print(
        "nombre de matières théoriques présentes dans le catalogue:",
        len(matières_théoriques)
    )

###############################################################################
# EXO 2
###############################################################################
def exercice2(promo):
    noms = [etu[1] for etu in promo]
    print(
        "Liste des noms d'étudiants dans l'ordre alphabétique:",
        " ".join(sorted(noms))
    )

###############################################################################
# EXO 3
###############################################################################
def DictionnaireApogee(catalogue):
    """cree et retourne un dictionnaire qui associe un code apogee à l'intitule
de l'UF"""
    mapping = {}
    for uf in catalogue:
        mapping[uf[0]] = uf[1]
    return mapping

def IntituleListeCodeApogee(catalogue, lstcode):
```

```python
    """affiche les intitules des matieres associés à la liste des codes apogee
lstcode"""
    map = DictionnaireApogee(catalogue)
    for code in lstcode:
        print(f"{code}: {map[code]}")

def exercice3(promo, catalogue):
    for etu in promo:
        print("Matières suivies par", etu[1])
        IntituleListeCodeApogee(catalogue, etu[3])

if __name__ == "__main__":
    catalogue=[
        ['KAEI7M01', 'Anglais 3', 'P', 30, 2],
        ['KAEI7M02', 'Communication 2', 'P', 15, 1],
        ['KAEI7M03', 'Mathematiques 3', 'T', 15, 1],
        ['KAEI7M04', 'Programmation 3', 'P', 30, 2],
        ['KAEI7M05', 'Informatique industrielle 1', 'P', 30, 2],
        ['KAEI7M06', 'Electronique 3', 'P', 60, 4],
        ['KAEI7M07', 'Traitement numerique des signaux 1', 'T', 30, 2],
        ['KAEI8M01', 'Anglais 4', 'P', 30, 2],
        ['KAEI8M02', 'Gestion de projets', 'P', 30, 2],
        ['KAEI8M03', 'Economie', 'T', 15, 1],
        ['KAEI8M04', 'Analyse numerique', 'T', 30, 2],
        ['KAEI8M05', 'UNIX 1', 'P', 15, 1],
        ['KAEI8M06', 'Informatique industrielle 2', 'P', 30, 2],
        ['KAEI8M07', "Conversion d'energie 2", 'T', 30, 2],
        ['KAEI8M08', 'Traitement numerique des signaux 2', 'T', 30, 2],
        ['KAEI8M09', 'Regulation numerique', 'T', 45, 3]
    ]

    promotion= [
        [190128003, 'Thomas', 7, ['KAEI7M01', 'KAEI7M02', 'KAEI7M03', 'KAEI7M04',
'KAEI7M05', 'KAEI7M06', 'KAEI7M07']],
        [190128004, 'Petit', 9, ['KAEI8M01', 'KAEI8M02', 'KAEI8M03', 'KAEI8M04',
'KAEI8M05', 'KAEI8M06', 'KAEI8M07', 'KAEI8M08', 'KAEI8M09']],
        [190128009, 'Aguilar', 7, ['KAEI7M01', 'KAEI7M02', 'KAEI7M03',
'KAEI7M04', 'KAEI7M05', 'KAEI7M06', 'KAEI7M07']],
        [190128010, 'Alvar', 9, ['KAEI8M01', 'KAEI8M02', 'KAEI8M03', 'KAEI8M04',
'KAEI8M05', 'KAEI8M06', 'KAEI8M07', 'KAEI8M08', 'KAEI8M09']],
        [190128015, 'Santos', 7, ['KAEI7M01', 'KAEI7M02', 'KAEI7M03', 'KAEI7M04',
'KAEI7M05', 'KAEI7M06', 'KAEI7M07']],
        [190128016, 'Pereira', 9, ['KAEI8M01', 'KAEI8M02', 'KAEI8M03',
'KAEI8M04', 'KAEI8M05', 'KAEI8M06', 'KAEI8M07', 'KAEI8M08', 'KAEI8M09']],
        [190128021, 'Peeters', 7, ['KAEI7M01', 'KAEI7M02', 'KAEI7M03',
'KAEI7M04', 'KAEI7M05', 'KAEI7M06', 'KAEI7M07']],
        [190128022, 'Jacobs', 9, ['KAEI8M01', 'KAEI8M02', 'KAEI8M03', 'KAEI8M04',
'KAEI8M05', 'KAEI8M06', 'KAEI8M07', 'KAEI8M08', 'KAEI8M09']]
    ]

    exercice1(catalogue)
    exercice2(promotion)
    exercice3(promotion, catalogue)
```

2. serie2.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
import os

def python_info():
    print("OS:", sys.platform)
    print("Interpreter path:", sys.executable)
    print("Python Version:", sys.version)

def listing(cwd='.', levels=0):
    # print wrapper that intents everything
    def iprint(*args):
        print("    "*levels, *args, sep="")

    for item in os.listdir(cwd):
        path = os.path.join(cwd, item)
        # if item is a directory, descend into it and list it's contents
        if os.path.isdir(path):
            iprint(item+"/")
            listing(path, levels+1)
        elif os.path.isfile(path):
            iprint(item)

if __name__ == "__main__":
    print("Python Info:")
    python_info()

    print("Directory listing:")
    listing('..')
```

3. serie3.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from sys import argv

def reverse_file(ifile):
    for line in ifile:
        line = line.rstrip('\n')
        yield line[::-1] + '\n'

def safe_int(data):
    try:
        return int(data)
    except (ValueError, TypeError):
        print(f"Error: '{data}' couldn't be parsed as an integer.")

months = [
    "janvier",
    "février",
    "mars",
    "avril",
    "mai",
    "juin",
    "juillet",
    "août",
    "septembre",
    "octobre",
    "novembre",
    "décembre"
]

def parse_flight_line(line, month):
    # Make sure that line is valid before parsing it
    split_line = [safe_int(num) for num in line.split()]
    if len(split_line) != 9:
        print("Syntax error in line")
        print("> ", line)
        raise SyntaxError

    data = {
        "flight_number" : split_line[0],
        "departure" : f"{split_line[1]:02} {months[month]:9} à 
{split_line[3]:02}:{split_line[4]:02}",
        "nb_seats" : split_line[7],
        "nb_passagers" : split_line[8]
    }
    # if we arrive "before" we leave, increment the month for the arrival
    if split_line[2] < split_line[1]:
        data["arrival"] = f"{split_line[2]:02} {months[(month+1)%13]:9} à 
{split_line[5]:02}:{split_line[6]:02}"
    else:
        data["arrival"] = f"{split_line[2]:02} {months[month]:9} à 
{split_line[5]:02}:{split_line[6]:02}"
    return data

def nb_passengers(flight_file):
    total_passagers = 0
    # ensure we're at the start of the file
    flight_file.seek(0, 0)
    # read the month
    month = flight_file.readline().rstrip('\n')
    try:
        month = months.index(month)
    except ValueError:
        print(f"Error: '{month}' is not a month")
        raise
    # read all following flight records from file
    for line in flight_file:
        parsed_line = parse_flight_line(line, month)
        total_passagers += parsed_line["nb_passagers"]
    return total_passagers

def humanize_flight_records(flight_file):
    # ensure we're at the start of the file
    flight_file.seek(0, 0)
    # read the month
    month = flight_file.readline().rstrip('\n')
    try:
```

```python
            month = months.index(month)
        except ValueError:
            print(f"Error: '{month}' is not a month")
            raise
        # read all following flight records from file
        for line in flight_file:
            parsed_line = parse_flight_line(line, month)
            print("Numéro du vol :", parsed_line["flight_number"])
            print("Départ le     :", parsed_line["departure"])
            print("Arrivée le    :", parsed_line["arrival"])

if __name__ == "__main__":
    if len(argv) != 3:
        print(f"Usage: {argv[0]} file_to_reverse flight_file")
        exit(1)

    to_mirror = argv[1]
    with open(to_mirror, 'r') as ifile, open(to_mirror+".mirrored", "w") as
ofile:
        ofile.writelines(reverse_file(ifile))

    flight_log = argv[2]
    with open(flight_log, "r") as ifile:
        print("Number of passengers :", nb_passengers(ifile))
        humanize_flight_records(ifile)
```