## Table des matières :

[EXERCICE 1](#_oqahslca0tq)

Méthode

Jeux d’essais

[EXERCICE 2](#_l977lb5lt7qv)

Méthode

Jeux d’essais

[EXERCICE 3](#_fgjlvic1qxqz)

Méthode

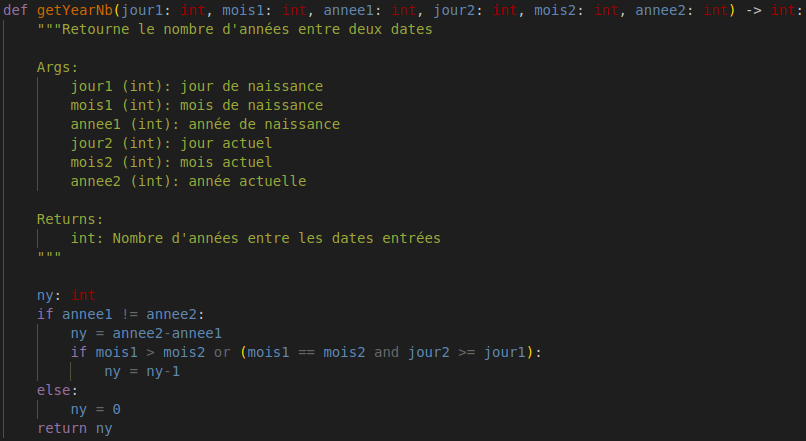
Jeux d’essais

## **EXERCICE 1 :**

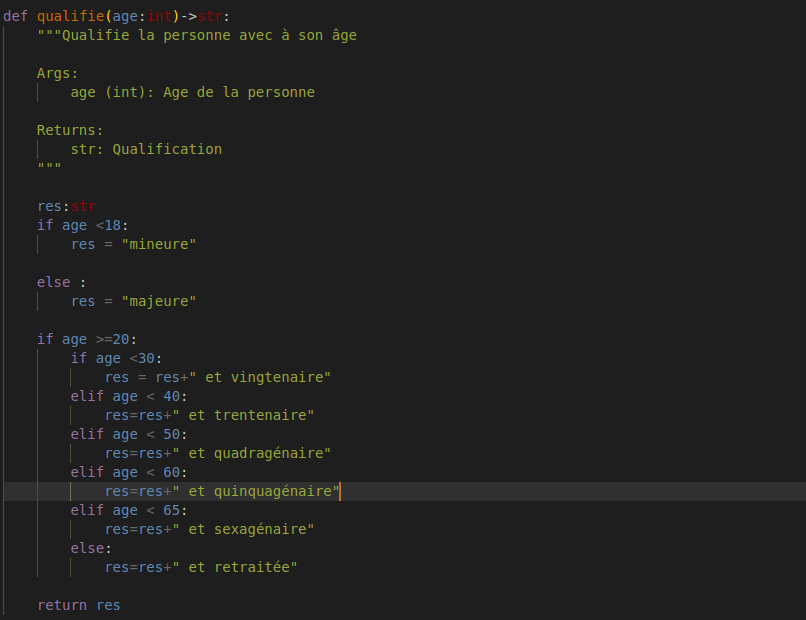
### **Méthode :**



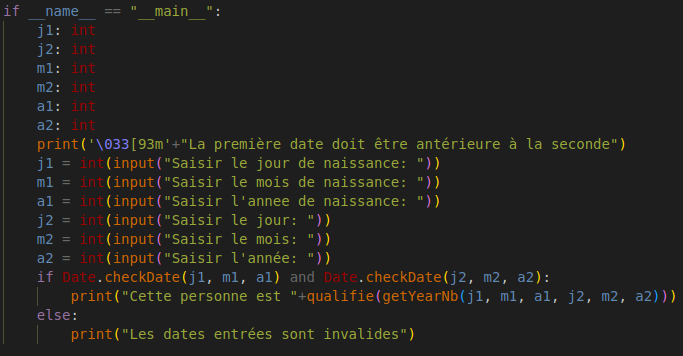
On importe de cette manière notre ancien programme contenant la fonction permettant de déterminer la validité d’une date.



On crée une première fonction, prenant en entrée les deux dates sous la forme de six entiers, qui vérifie combien d'années se sont écoulées entre ces deux dates et retourne ce résultat sous la forme d’un entier.

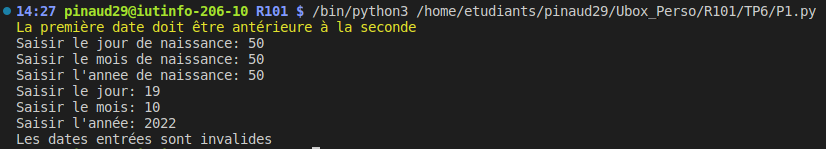


On définit une seconde fonction qui vérifie en fonction de la valeur de l’entier en entrée la qualification de la personne grâce à une suite de conditions. On vérifie d’abord si la personne est majeure ou mineure, puis si l’on doit ajouter d’autres qualificatifs.

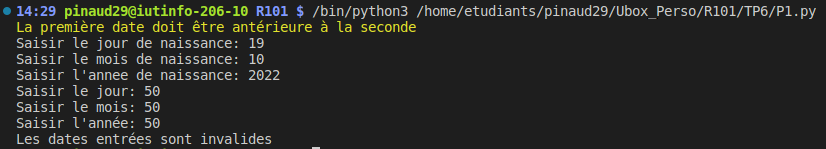


On crée enfin un programme principal qui demande les valeurs à l’utilisateur et utilise les deux fonctions après vérification de la date par la fonction checkDate présente dans Date. On utilise un print pour avertir l’utilisateur que la première date doit être antérieure à la seconde.

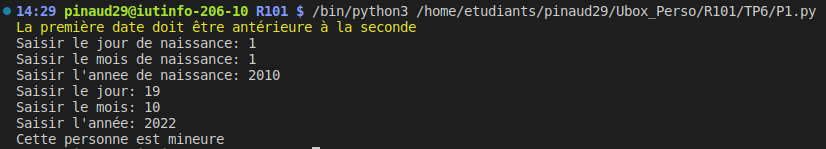
### **Jeux d’essais :**



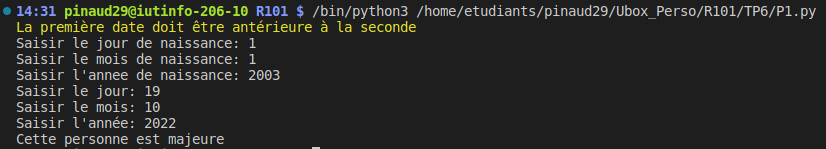
Avec la 1ere date invalide



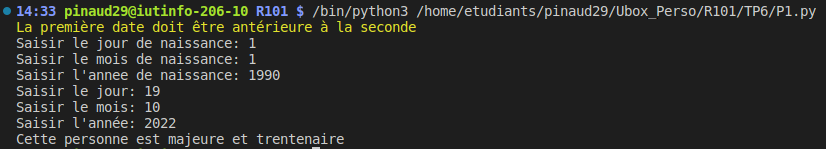
Avec la 2eme date invalide



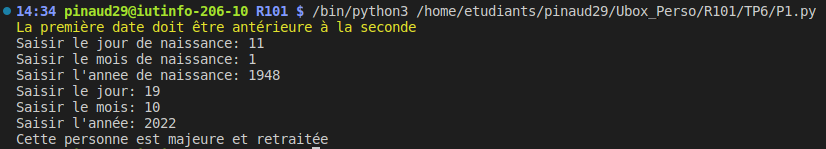
Personne mineure sans attribut supplémentaire



Personne majeure sans attribut supplémentaire



Personne majeure avec un attribut supplémentaire



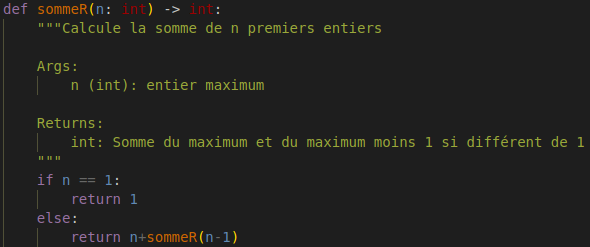
Personne majeure avec un âge supérieur à 65 ans

## **EXERCICE 2 :**

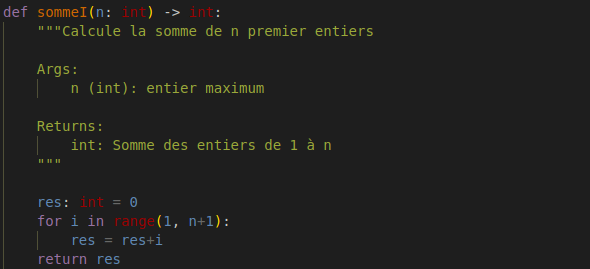
### **Méthode :**



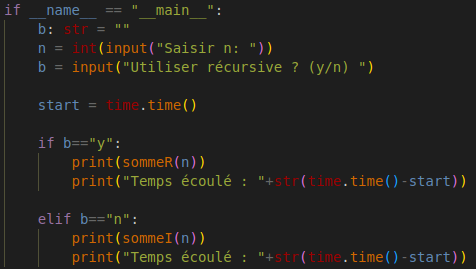
On importe time pour pouvoir chronométrer les deux algorithmes



On crée une version récursive de l’algorithme dans une première fonction

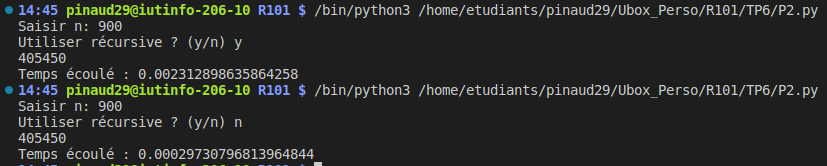


Puis une version itérative effectuant exactement le même calcul.

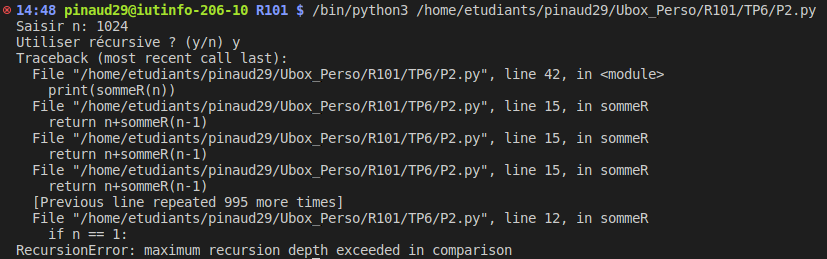


On crée un programme demandant à l’utilisateur de choisir parmi les deux algorithmes puis on exécute celui choisi avec la valeur n en entrée (elle aussi choisie par l’utilisateur). Le programme affiche le résultat de la somme ainsi que le temps pris pour la calculer.

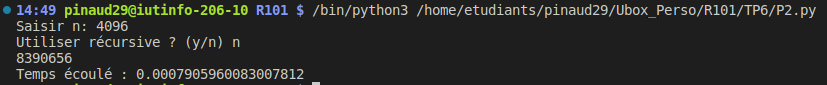
### **Jeux d’essais :**



On utilise chaque algo avec 900 en valeur d’entrée. On peut comparer les temps, qui semblent plus courts dans le cas d’une utilisation itérative. Cependant, les calculs sont exécutés très rapidement donc cette différence pourrait ne pas être significative.



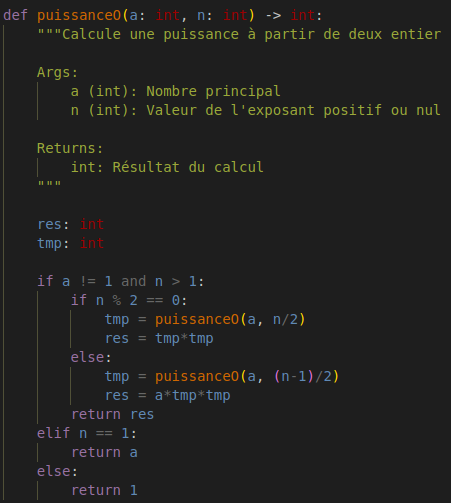
Cependant, on peut voir qu’à partir de certaines valeurs, la fonction récursive renvoie une erreur, elle est donc rapidement limitée.



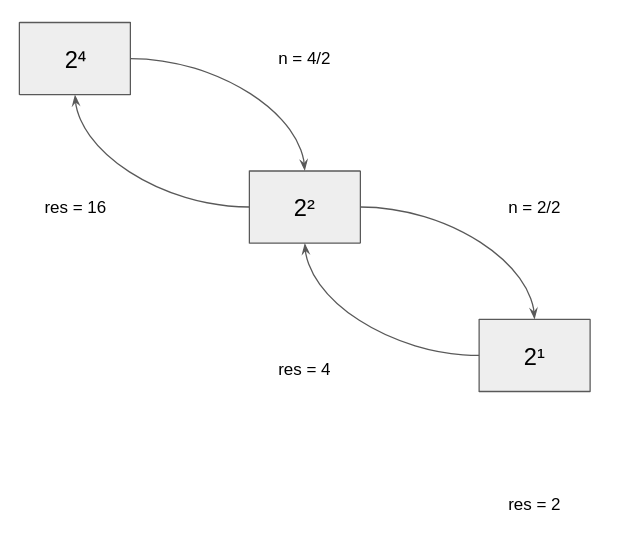
Ce qui n’est pas le cas de la version itérative, qui effectue le calcul sans problème.

## **EXERCICE 3 :**

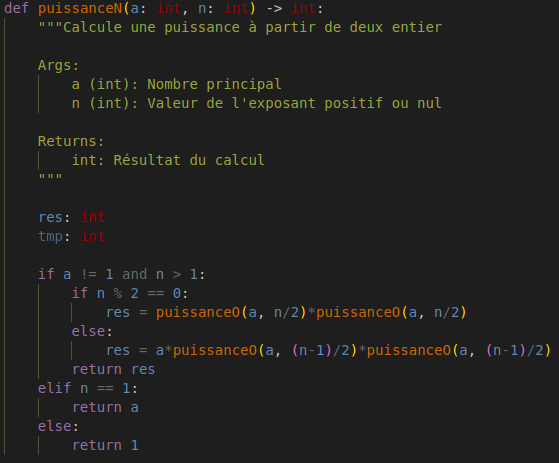
### **Méthode :**



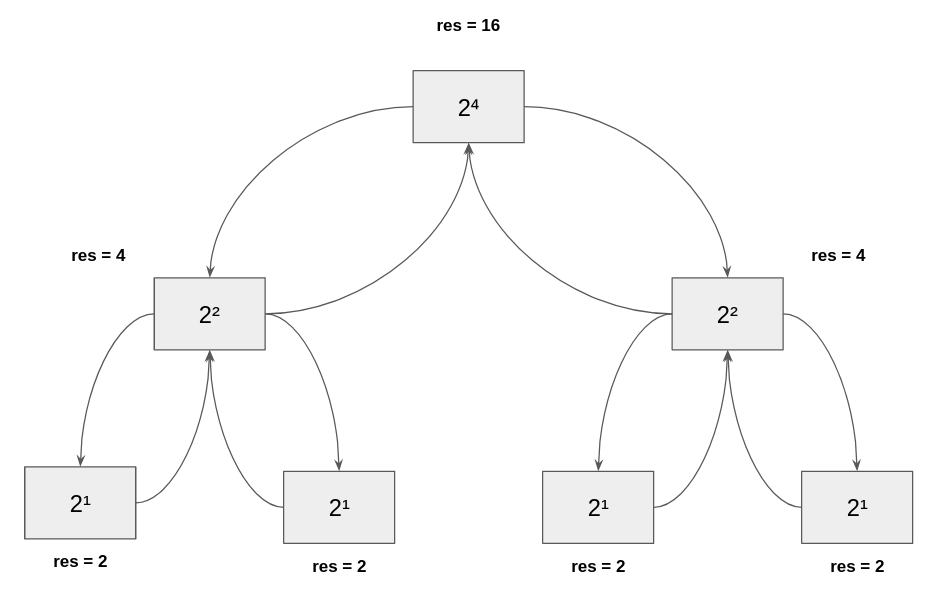
On crée une nouvelle fonction correspondant à la version optimisée de la fonction, elle n’effectue qu’une seule “branche” de récursivité.



*Pour a=2 et n=4, optimisé*

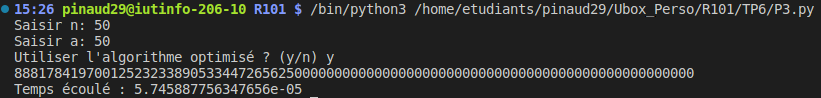


Et l’on crée une version non optimisée utilisant 2 fois la récursivité.

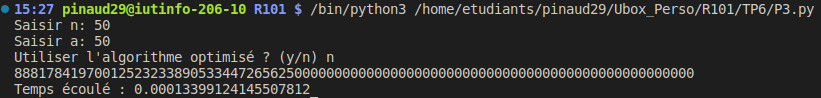


pour a=2 et n=4, non optimisé

### **Jeux d’essais :**

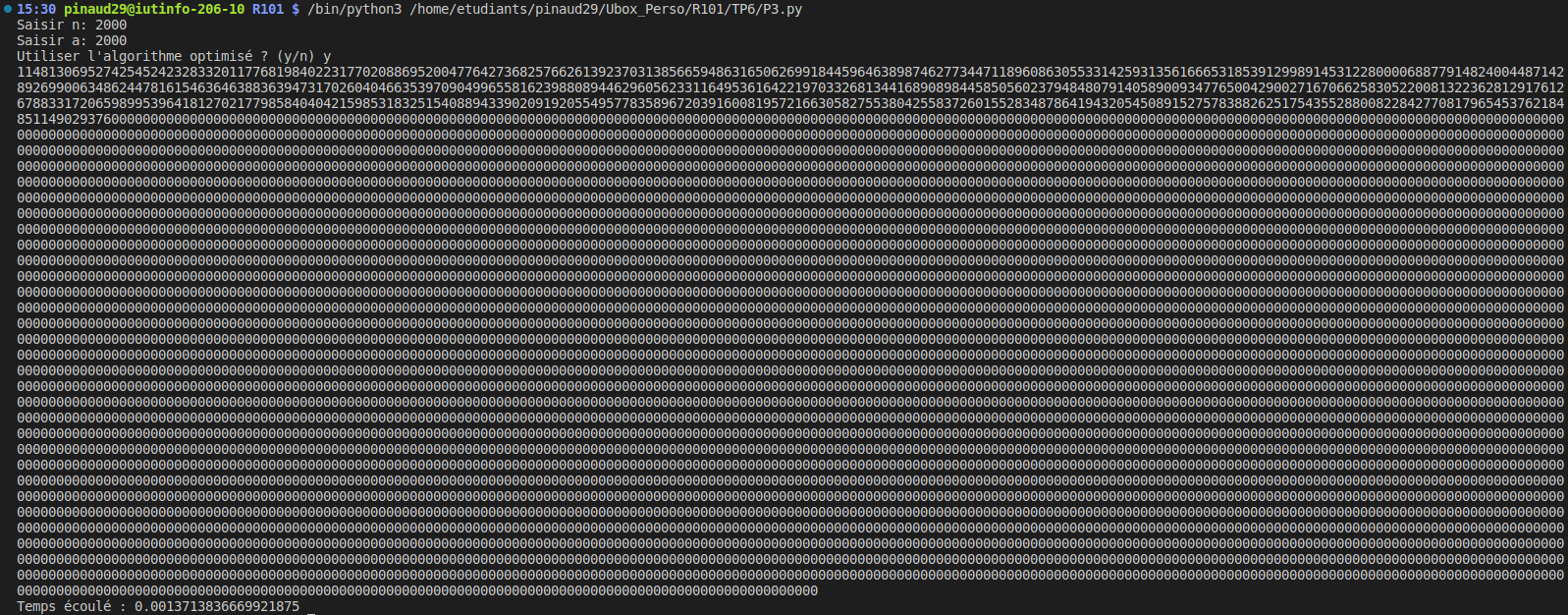


Avec l’algo optimisé



Avec l’algo non-optimisé.

La différence de temps n’est pas significative, elle change souvent au cours des tests et ne semble pas correspondre, cependant, il suffit de regarder l'arborescence des schémas pour constater l’importance de cette optimisation au niveau des performances et de l’utilisation de mémoire.



On note que l’on peut aller très haut dans les valeurs choisies.