Épisode IX : Révisions et compléments

EXERCICE 1

Écrire une fonction `uneMinuteEnPlus` qui calcule et retourne l'heure une minute après celle passée en paramètre sous forme de deux entiers que l'on suppose cohérents. Exemples :

- `uneMinuteEnPlus(14,32)` retourne (14, 33).
- `uneMinuteEnPlus(14,59)` retourne (15, 0).

Ne pas oublier le cas de minuit.

EXERCICE 2

Rappel : On dit que i est un diviseur de n si le reste de la division de n par i est égal à 0.

1. Écrire une fonction `estDiviseur(i,n)` qui retourne `True` si `i` est un diviseur de `n` et `False` sinon.
2. Un nombre est dit *premier* s'il n'a que 2 diviseurs : 1 et lui-même. Calculez à la main la liste des nombres premiers inférieurs à 15.
3. Écrire une fonction `estPremier(n)` qui retourne `True` si `n` est premier, `False` sinon (on profitera du fait que seuls les nombres strictement inférieurs à `n` peuvent être diviseurs de `n`).
4. Écrire une fonction `nbPremiers(n)` qui retourne le nombre de nombres premiers strictement plus petits que n.

EXERCICE 3

L'objectif est d'écrire un *test de primalité*, c'est à dire une fonction `premier(n)` qui retourne `True` si l'entier naturel $n > 1$ est premier et `False` sinon. Pour cela on parcourt les nombres supérieurs à 2 pour chercher un diviseur d de n : dès que l'on en trouve un on arrête les calculs, n n'est pas premier.

1. Si on ne trouve pas de diviseur $d \leq \sqrt{n}$, on est sûr que n est premier : pourquoi ? Comment effectuer le test $d \leq \sqrt{n}$ sans utiliser de fonction « racine carrée » ?
2. Écrire la fonction `premier(n)` ; tester cette fonction en affichant tous les nombres premiers inférieurs à 100.
3. Améliorer[a] `premier(n)` en traitant à part le cas où n est pair ; dans le cas où n est impair, il suffit ensuite de chercher un diviseur d impair.

a. il existe des tests de primalité sophistiqués *radicalement* plus efficaces que le test naïf décrit dans l'exercice 3 ; ils permettent de tester en quelques secondes la primalité d'un nombre dont l'écriture décimale comporte plusieurs centaines de chiffres.

EXERCICE 4

Nous vous fournissons un module `bibcsv.py` qui permet d'ouvrir des fichiers CSV contenant juste une liste de nombres. Ce module est disponible en téléchargement ici. Utiliser un clic droit et "enregistrer sous" pour l'enregistrer à côté de vos autres fichiers python. Pour utiliser un module il faut commencer par l'*importer*, et toute session de travail utilisant ce module doit commencer par la phrase magique :

```
from bibcsv import *
```

Vous disposez alors de la fonction

`ouvrirCSV(nom)`	Ouvre le fichier *nom* et retourne la liste de nombres qu'il contient (par exemple `ouvrirCSV("notes.csv")`).

Récupérer le fichier `notes.csv` depuis le site du cours, l'enregistrer de la même façon, et utiliser `ouvrirCSV` pour récupérer la liste des nombres stockée dans le fichier CSV :

```
maliste = ouvrirCSV("notes.csv")
```

et observer le contenu de la variable `maliste`.

1. Utilisez les fonctions `moyenneListe`, `ecartTypeListe`, `maximumListe`, pour analyser la liste de notes contenue dans `maliste`.
2. Vous pouvez créer votre propre fichier `.csv` avec LibreOffice. Dans une feuille de calcul, mettez les nombres à la suite dans la première colonne uniquement (ou bien en les copiant/collant depuis un document existant). Utilisez "Fichier", "Enregistrer sous", saisissez un nom de fichier en utilisant l'extension `.csv` et validez, confirmez que c'est bien le format CSV que vous désirez utiliser, et utilisez les options par défaut. Vous pouvez alors charger le fichier dans python à l'aide d'`ouvrirCSV` et effectuer les mêmes analyses.
3. Récupérez sur le site et ouvrez de la même façon le fichier `temperatures.csv` , effectuez les mêmes analyses.
4. Observez la fonction suivante :

```
def mystere(L, x):
    cpt = 0
    for i in L :
        if i > x:
            cpt = cpt + 1
            if cpt == 3:
                return True
        else:
            cpt = 0
    return False
```

   Que retourne-t-elle lorsqu'on lui passe en paramètres la liste des températures et 30 ? Et lorsque l'on passe 35 au lieu de 30 ? Faites-la tourner dans Spyder pour bien comprendre ce qui se passe.

EXERCICE 5

1. Écrire une fonction `facteurImpair(n)`, qui, étant donné un entier naturel `n` non-nul, renvoie le plus grand diviseur impair de `n`. Le résultat sera obtenu par une succession de divisions de `n` par 2 tant que `n` est pair. Par exemple, `facteurImpair(504)` retournera 63, puisque 504 est pair, 252 (504 divisé par 2) et 126 $(252//2)$ sont pairs, et 63 $(126//2)$ est impair.
2. Écrire une fonction `puissanceDiviseur(p,n)`, qui, étant donné un entier naturel `n` non-nul, renvoie la plus grande puissance de `p` qui divise `n`.

   Par exemple, `puissanceDiviseur(2,504)` retournera `8`, puisque 504 est divisible par $8 = 2^3$, mais pas par $16 = 2^4$.
3. Écrire une fonction `premierSuivant(n)` qui renvoie le premier nombre premier strictement supérieur à `n`. Par exemple, `premierSuivant(2)` vaut 3, `premierSuivant(3)` et `premierSuivant(4)` valent 5.
4. En utilisant la fonction `premierSuivant(n)`, écrire une fonction `decompositionFacteursPremiers(n)` qui renvoie la liste des nombres constituant la décomposition de `n` en un produit de facteurs premiers.

EXERCICE 6

Une suite de Syracuse est définie par récurrence de la façon suivante :

$$\begin{cases} u_0 & = a \\ u_{n+1} & = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{sinon.} \end{cases} \end{cases}$$

où a est un entier naturel strictement positif.

1. Calculer à la main les premiers termes de la suite pour $a = 5$ puis pour $a = 7$.
2. Écrire une fonction `syracuse(a, n)` qui calcule le terme de rang n de cette suite lorsque le premier terme u_0 est égal à a. Tester cette fonction pour $a = 5$ puis pour $a = 7$.
3. Que se passe-t-il lorsque pour une valeur de n, u_n est égal à 1 ?
4. On conjecture (consulter par exemple Wikipedia) qu'une suite de Syracuse finit toujours par atteindre la valeur 1. Écrire une fonction `longueur(a)` qui calcule et retourne la première valeur de n telle que $u_n = 1$ lorsque le premier terme u_0 vaut a.
5. Vérifier la conjecture pour tous les entiers $a < 100$ (utilisez une boucle bien sûr, en utilisant `print` pour montrer les résultats !) Parmi ces valeurs de a, quelle est celle qui fournit une suite de longueur maximale ?
6. Utiliser la fonction *syracuse* de la question 2 pour écrire la fonction *longueur* de la question 4 est une idée naturelle. Etudier dans ce cas combien de fois on exécute le calcul :

   $$u_{n+1} = u_n/2 \quad \text{si } u_n \text{ est pair,} \quad u_{n+1} = 3u_n + 1 \quad \text{sinon,}$$

   pour calculer longueur(27). Améliorer le code de la fonction `longueur(a)` pour éviter les calculs inutiles.
7. Écrire la fonction `listeSyracuse(a)` qui calcule et retourne la *liste*

   $$[u_0 = a, u_1, u_2, ..., u_n = 1]$$

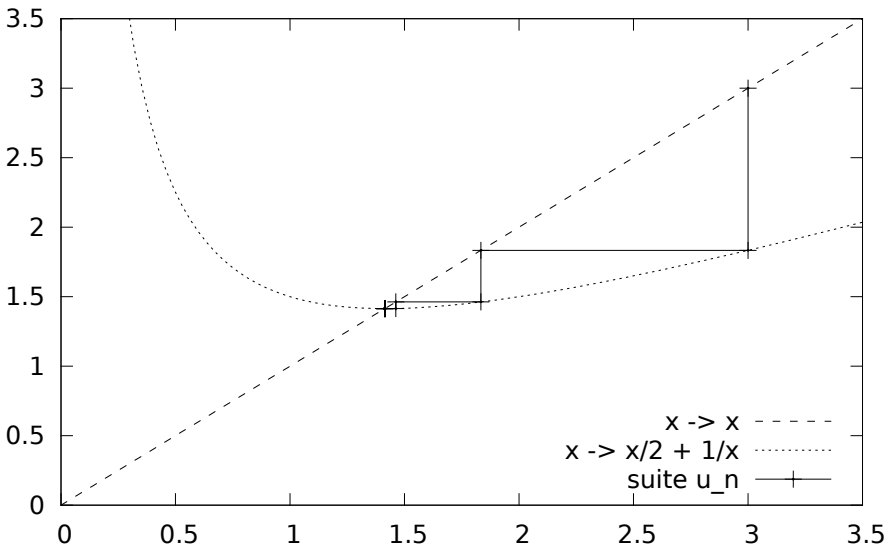
   où u_n désigne le premier terme égal à 1. Par exemple, avec $a = 7$ on obtient la liste [7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1].
8. Écrire une fonction `hauteur(a)` qui calcule et retourne la valeur maximale de u_n lorsque le premier terme u_0 vaut a ; par exemple `hauteur(7)` vaut 52.

EXERCICE 7

Comment calculer $\sqrt{2}$ avec k chiffres après la virgule ? Une méthode simple est d'utiliser la suite u_n suivante :

$$\begin{cases} u_0 & = 3 \\ u_{n+1} & = \frac{u_n}{2} + \frac{1}{u_n} \end{cases}$$

On peut notamment dessiner le graphe suivant pour observer la convergence.



1. Écrire une fonction `suite(n)` qui calcule et retourne le terme de rang n de cette suite.
2. Utilisez une boucle pour afficher les 10 premiers termes de la suite. Une valeur approchée de $\sqrt{2}$ est $1.41421356237309504[...]$. On constate effectivement que la suite converge vers cette valeur, mais sans pouvoir toutefois l'atteindre : le nombre de chiffres des valeurs de l'ordinateur est limité !
3. Plus généralement, pour calculer $\sqrt{x}$, on peut utiliser la suite

$$\begin{cases} u_0 & = x \\ u_{n+1} & = \frac{1}{2}(u_n + \frac{x}{u_n}) \end{cases}$$

   où u_0 est une première estimation grossière de $\sqrt{x}$, on a utilisé x lui-même pour simplifier. Écrire une fonction `sqrt(x)` qui retourne une approximation de $\sqrt{x}$ en calculant u_{10} et imprimant les valeurs intermédiaires u_n au passage. Tester avec 2, 3, 10.
4. Tester avec 1000. On constate que la convergence est lente, on n'est pas sûr que 10 itérations suffisent. Remplacer la boucle `for` par une boucle `while` pour continuer le calcul tant que la différence entre deux termes consécutifs est plus grande que 0.0000001. Tester avec 100000.