IFOSUP

Institut de Formation Supérieure
Ville de Wavre

Algorithmique: Exercices

Cédric Vanconingsloo

Table des matières

1. Python interactif

1.1. Prédire les opérations

Prédisez le résultat de chacune de ces opérations, puis vérifiez avec l'interpréteur.

- (1+2) ** 3
- "Da" * 4
- "Da" + 4
- ("Pa" + "La") * 2
- ("Da" * 4) / 2
- 5 / 2
- 5 // 2
- 5 % 2

1.2. Prédire les opérations et transtypages

Prédisez le résultat de chacune de ces opérations, puis vérifiez avec l'interpréteur.

- str(4) * int("3")
- int("3") + float("3.2")
- str(3) * float("3.2")
- str(3/4) * 2

1.3. Afficher du texte

Utilisez l'interpréteur Python pour afficher les textes suivants:

- Hello World
- Aujourd'hui
- C'est "dommage"!
- Hum \0/

2. Les variables

2.1. Échange

Écrivez un algorithme permettant d'échanger les valeurs de deux variables **a** et **b**, et ce *quel que soit le contenu préalable*.

2.2. Transfert

Écrivez un algorithme permettant de transférer les valeurs de trois variables **a**, **b** et **c**. Il faut transférer **a** dans **b**, **b** dans **c** et **c** dans **a**, *quels que soient les contenus préalables de ces variables*.

2.3. Carré

Écrivez un algorithme qui demande un nombre à l'utilisateur, puis qui calcule *le carré de ce nombre*.

2.4. Bonjour

Écrivez un algorithme qui demande le prénom à l'utilisateur, puis qui affiche **Bonjour *utilisateur*!**

2.5. Magasin

Écrivez un algorithme qui lit le nom d'un article, son prix hors taxe, le nombre d'articles et le taux de TVA, puis qui fournit *le prix total TTC correspondant*. Faites en sorte que l'algorithme pose des questions claire et affiche un résultat complet.

3. Opérateurs

3.1. Test 1

Écrivez un algorithme qui réponde *True* quand un nombre donné par l'utilisateur est *pair*.

3.2. Test 2

Écrivez un algorithme qui réponde *True* quand un nombre donné par l'utilisateur est *positif*.

3.3. Test 3

Écrivez un algorithme qui réponde *True* quand un nombre donné par l'utilisateur est *plus petit que 50*.

3.4. Test 4

Écrivez un algorithme qui réponde *True* quand un nombre donné par l'utilisateur est *plus grand que 50*.

3.5. Test 5

Écrivez un algorithme qui réponde *True* quand un nombre donné par l'utilisateur est *entre 10 et 100*.

3.6. Test 6

Écrivez un algorithme qui réponde *True* quand un nombre donné par l'utilisateur est *divisible par 3*.

3.7. Test 7

Écrivez un algorithme qui réponde *True* quand un nombre donné par l'utilisateur est *impair* ***ou*** *divisible par 5*.

3.8. Test 8

Écrivez un algorithme qui réponde *True* quand un nombre donné par l'utilisateur est *pair* ***et*** *divisible par 5*.

3.9. Test 9

Écrivez un algorithme qui réponde *True* quand un **mot** donné par l'utilisateur est ***avant*** *le mot* combinatoire *dans le dictionnaire*.

3.10. Test 10

Écrivez un algorithme qui réponde *True* quand un **mot** donné par l'utilisateur est ***après*** *le mot* combinatoire *et* ***avant*** *le mot* logique *dans le dictionnaire*.

4. Tests et conditions

4.1. Positif ou Négatif 1

Écrivez un algorithme qui demande un nombre à l'utilisateur, puis qui l'informe *si ce nombre est positif ou négatif*. On considère que **0** est positif.

4.2. Produit positif ou négatif

Écrivez un algorithme qui demande deux nombres à l'utilisateur, puis qui l'informe *si leur produit est positif ou négatif*. On laisse de côté le cas où le produit est **nul**. **Attention, il ne faut pas calculer le produit!!**

4.3. Dico

Écrivez un algorithme qui demande **trois mots** à l'utilisateur, puis qui l'informe *s'ils sont rangés ou non dans l'ordre alphabétique*. **Attention, il ne faut pas les trier!**

4.4. Positif ou Négatif 2

Écrivez un algorithme qui demande un nombre à l'utilisateur, puis qui l'informe *si ce nombre est* ***positif***, ***négatif*** *ou* ***nul***.

4.5. Produit positif ou négatif 2

Écrivez un algorithme qui demande deux nombres à l'utilisateur, puis qui l'informe *si leur produit est* ***positif***, ***négatif*** *ou* ***nul***. **Attention, il ne faut pas calculer le produit**.

4.6. Catégorie

Écrivez un algorithme qui demande l'âge d'un enfant à l'utilisateur, puis qui l'informe de sa catégorie:
Poussin de 6 à 7 ans;
Pupille de 8 à 9 ans;
Minime de 10 à 11 ans;
Cadet de 12 à 15 ans;
Espoir après 15 ans.

4.7. Maximum

Écrivez un algorithme qui demande **trois nombres** à l'utilisateur, puis qui *affiche le nombre maximum*.

4.8. Voyelles

Écrivez un algorithme qui demande **une lettre** à l'utilisateur, puis qui l'informe *s'il s'agit d'une voyelle ou d'une consonne*.

4.9. Jour

Écrivez un algorithme qui demande un nombre entre **1** et **7** à l'utilisateur, puis qui affiche *le jour de la semaine correspondant*.

5. Logique

5.1. Avenir 1

Écrivez un algorithme capable de prédire l'avenir. Demandez à l'utilisateur de rentrer l'heure puis les minutes. L'algorithme indiquera l'heure qu'il sera **une minute plus tard**. On suppose que l'utilisateur entre une heure valide.

5.2. Avenir 2

Écrivez un algorithme similaire au précédent, mais en gérant les secondes. L'algorithme affichera l'heure qu'il sera **une seconde plus tard**.

5.3. Facturation

Un magasin de photocopies facture 0,05€ les 10 premières photocopies, 0,04€ les 20 suivantes et 0,03€ au-delà. Écrivez un algorithme qui demande à l'utilisateur le nombre de photocopies effectuées et qui affiche la facture correspondante.

5.4. Impôt

Les habitants de Lenclume paient l'impôt selon les règles suivantes:

- Les hommes de plus de 20 ans paient l'impôt;
- Les femmes paient l'impôt si elles ont entre 18 et 35 ans;
- Les autres ne paient pas d'impôts.

Écrivez un algorithme qui demande l'âge et le sexe de l'Argonien, puis qui indiquera s'il doit payer un impôt ou non.

5.5. Élections

Les élections législatives en Bordeciel obéissent à la règle suivante:

- Lorsqu'un des candidats obtient plus de 50% des suffrages, il est élu dès le premier tour.
- En cas de second tour, seuls les candidats ayant obtenu au moins 12,5% des voix au premier tour peuvent participer.

Écrivez un algorithme qui permette la saisie des scores de quatre candidats au premier tour. Cet algorithme traitera ensuite **uniquement** le premier candidat. Il dira s'il est élu, battu, en ballottage favorable (il participe au second tour en étant arrivé en tête à l'issue du premier tour), ou en ballottage défavorable (il participe au second tour sans avoir été en tête au premier tour).

5.6. Assurance

Une compagnie d'assurance automobile propose à ses clients quatre sortes de tarifs identifiables par une couleur, du moins au plus onéreux: le tarif bleu, le tarif vert, le tarif orange et le tarif rouge. Le tarif dépend de la situation du conducteur:

1. Un conducteur de moins de 25 ans et titulaire du permis depuis moins de deux ans, se voit attribuer le tarif rouge, si toutefois il n'a jamais été responsable d'accident. Sinon, la compagnie **refuse de l'assurer**.
2. Un conducteur de moins de 25 ans et titulaire du permis depuis plus de deux ans, ou de plus de 25 ans, mais titulaire du permis depuis moins de deux ans a droit au tarif orange s'il n'a jamais provoqué d'accident, au tarif rouge s'il a eu un accident, sinon il est **refusé**.

3. Un conducteur de plus de 25 ans titulaire du permis depuis plus de deux ans bénéficie du tarif vert s'il n'est à l'origine d'aucun accident, du tarif orange s'il a eu un accident, du rouge s'il en a eu deux, et sera **refusé** au-delà.
4. De plus, pour encourager la fidélité des clients acceptés, la compagnie propose un contrat de la couleur immédiatement la plus avantageuse s'il est client de la compagnie depuis plus de 5 ans. Ainsi, s'il satisfait cette exigence, un client normalement vert deviendra bleu, un client normalement orange devient vert et le rouge devient orange.

Écrivez un algorithme permettant de saisir les données nécessaires (sans contrôle de saisie) et de traiter le problème.

5.7. Validité de date

Écrivez un algorithme qui demande à l'utilisateur un numéro de jour, de mois et d'année, puis qui renvoie s'il s'agit ou non d'une date valide. Pour rappel, une année est *bissextile* si elle est divisible par 4. Les années divisibles par **100** ne sont pas bissextiles, mais les années divisibles par 400 le sont.

5.8. Équation quadratique

Écrivez un algorithme qui permet de trouver toutes les racines d'une équation quadratique. En algèbre, une équation **quadratique** est une équation sous la forme $ax^2 + bx + c = 0$. Une équation quadratique peut avoir une ou deux racines réelles ou complexes distinctes selon la nature du **discriminant** de l'équation. Le **discriminant** de l'équation quadratique est donné par $\Delta = b^2 - 4ac$.

Selon la nature de discriminant, la formule de recherche des racines est donnée par:

- cas 1: Si le discriminant Δ est **positif**, il y a deux racines réelles distinctes données par $-b + \frac{\sqrt{\Delta}}{2a}$ et $-b - \frac{\sqrt{\Delta}}{2a}$.
- cas 2: Si le discriminant Δ est **nul**, il y a exactement une racine réelle donnée par $-\frac{b}{2a}$
- cas 3: Si le discriminant Δ est **négatif**, alors il y a deux racines complexes distinctes données par: $-\frac{b}{2a} + i\frac{\sqrt{-\Delta}}{2a}$ et $-\frac{b}{2a} - i\frac{\sqrt{-\Delta}}{2a}$

L'algorithme va demander les paramètres **a,b et c** avant d'afficher les racines quadratique. Pour ajouter la fonction `sqrt`, écrivez en première ligne `from math import sqrt`.

6. Les boucles

6.1. Entre 1 et 3

Écrivez un algorithme qui demande à l'utilisateur un nombre compris entre **1** et **3** jusqu'à ce que la réponse convienne.

6.2. Entre 10 et 20

Écrivez un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message: **Plus petit!**, et inversement **Plus grand!** si le nombre est inférieur à 10.

6.3. Nombres suivants 1

Écrivez un algorithme qui demande un nombre de départ, puis qui affiche ensuite les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre **33**, l'algorithme affichera les nombres **34** à **43**.

6.4. Nombres suivants 2

Modifiez l'algorithme précédent en utilisant une autre structure algorithmique de boucle.

6.5. Table de multiplication

Écrivez un algorithme qui demande un nombre de départ, puis écrit la table de multiplication de ce nombre, présentée comme suit:

```
Table de 8:
8 × 1 = 8
8 × 2 = 16
...
8 × 10 = 80
```

6.6. Somme

Écrivez un algorithme qui demande un nombre de départ, puis qui calcule la somme des entiers jusqu'à ce nombre. On souhaite afficher uniquement le résultat. Exemple avec le nombre 5: $1 + 2 + 3 + 4 + 5 = 15$

6.7. Factorielle

Écrivez un algorithme qui demande un nombre de départ, puis qui calcule la factorielle. Exemple avec le nombre 5: $1 * 2 * 3 * 4 * 5 = 120$

6.8. Plus grand nombre 1

Écrivez un algorithme qui demande successivement 20 nombres à l'utilisateur, puis qui lui indique ensuite quel est le plus grand parmi ces 20 nombres.

6.9. Plus grand nombre 2

Modifiez l'algorithme précédent pour qu'il affiche en plus en quelle position avait été saisie ce nombre.

6.10. Plus grand nombre 3

Modifiez l'algorithme précédent, mais cette fois-ci on ne connaît pas à l'avance combien l'utilisateur souhaite entrer de nombres. La saisie des nombres s'arrête lorsque l'utilisateur entre un **0**.

6.11. Décalage de date

Écrivez un algorithme qui demande une date **valide** à l'utilisateur, puis qui demande un nombre de jours. L'algorithme affichera alors la nouvelle date décalée. Ex : En entrant la première date `01/03/2025` suivi du décalage `45`, l'algorithme affichera `15/04/2025`.

6.12. Retour de monnaie

Écrivez un algorithme qui lit une suite de prix en euros entiers, terminée par un **0**, correspondant aux achats d'un client. Calculer la somme que le client doit payer, lire la somme qu'il paie et simuler la remise de la monnaie en affichant les textes **billets de 10€: ...**, ... autant de fois qu'il y a de coupures à rendre.

6.13. Tiercé

Écrivez un algorithme qui permette de connaître les chances de gagner au Tiercé, Quarté et Quinté. On demande à l'utilisateur le nombre de chevaux partants et le nombre de chevaux joués. Les deux messages affichés devront être:

```
Dans l'ordre: une chance sur X de gagner
Dans le désordre: une chance sur Y de gagner
```

X et **Y** nous sont donnés par les formules suivantes, si n est le nombre de chevaux partants et p le nombre de chevaux joués. Pour rappel, le signe ! signifie *factorielle* en mathématique.
$X = \frac{n!}{(n-p)!}$ $Y = \frac{n!}{p!*(n-p)!}$

6.14. Syracuse 1

Écrivez un algorithme qui affiche la conjecture de Syracuse. La conjecture de Syracuse est une suite d'entiers naturels définie de la manière suivante:

- On part d'un nombre entier plus grand que zéro;
- Si ce nombre est pair, on le divise par deux;
- Si ce nombre est impair, on le multiplie par trois et on ajoute un.

En répétant l'opération, on obtient une suite d'entiers positifs qui va finir par arriver à 1, avant de démarrer un cycle (1,4,2,1...)

6.15. Syracuse 2

Modifiez l'algorithme précédent pour ajouter différentes informations:

- le temps de vol: c'est le nombre d'éléments de la suite jusqu'à ce que la suite atteigne **1**.
- Le temps de vol en altitude: c'est le nombre d'éléments de la suite **au-dessus du nombre de base**.
- l'altitude maximale: c'est la valeur **maximale** de la suite.

7. Les tableaux simples

7.1. Tableau 1

Écrivez un algorithme qui déclare et remplisse un tableau de **7** valeurs numériques en les mettant toutes à *0*.

7.2. Voyelles

Écrivez un algorithme qui déclare et remplisse un tableau contenant les 6 voyelles de l'alphabet latin.

7.3. Notes

Écrivez un algorithme qui déclare un tableau de **9** notes, dont on fait ensuite saisir les valeurs par l'utilisateur.

7.4. Notes 2

Modifiez l'algorithme **Notes** afin d'effectuer et d'afficher le calcul de la moyenne des notes.

7.5. Tableau positif et négatif

Écrivez un algorithme permettant à l'utilisateur de saisir un nombre quelconque de valeurs qui devront être stockés dans un tableau. L'utilisateur doit donc commencer par entrer le nombre de valeurs qu'il compte saisir. Il effectuera ensuite cette saisie. Enfin, une fois la saisie terminée, le programme affichera le nombre de valeurs négatives et le nombre de valeurs positives.

7.6. Somme tableau

Écrivez un algorithme calculant la somme des valeurs d'un tableau préalablement saisi.

7.7. Somme tableaux

Écrivez un algorithme constituant un tableau à partir des deux tableaux suivants. Le nouveau tableau contiendra la somme des éléments des deux tableaux de départ.

4	8	7	9	1	5	4	6

7	6	5	2	1	3	7	4

11	14	12	11	2	8	11	10

7.8. Schtroumpf de tableau

À partir des deux tableaux précédents, écrivez un algorithme qui calcule le *Schtroumpf* des deux tableaux. Pour calculer le *Schtroumpf*, il faut multiplier chaque élément du tableau 1 par chaque élément du tableau 2 et additionner le tout.

7.9. Augmentation

Écrivez un algorithme qui permette la saisie d'un nombre quelconque de valeurs, sur le principe de l'exercice **tableau positif et négatif**. Toutes les valeurs doivent être ensuite augmentées de **1** et le nouveau tableau sera affiché à l'écran.

7.10. Plus grande valeur

Écrivez un algorithme permettant, toujours sur le même principe, à l'utilisateur de saisir un nombre déterminé de valeurs. L'algorithme, une fois a saisie terminée, renvoie la plus grande valeur en précisant quelle position elle occupe dans le tableau. On prendra soin d'effectuer la saisie dans un premier temps et la recherche de la plus grande valeur du tableau dans un second temps.

7.11. Moyenne de classe

Écrivez un algorithme permettant à l'utilisateur de saisir les notes d'une classe. L'algorithme, une fois la saisie terminée, renvoie le nombre de ces notes supérieures à la moyenne **de la classe**.

7.12. Somme Diviseur

Écrivez un algorithme permettant de stocker dans une liste tous les **entiers positifs de 3 chiffres** tel que, pour chaque entier, *la somme des chiffres* est un *diviseur du produit de ces chiffres*. Ex: 514 est un nombre valide, car $5 + 1 + 4 = 10$ est un diviseur de $5 \times 1 \times 4 = 20$.

8. Les tableaux à plusieurs dimensions

8.1. Tableau 2D 1

Écrivez un algorithme remplissant un tableau de 6x13 cases, avec des **0**.

8.2. Recherche 2D

Écrivez un algorithme qui recherche la plus grande valeur au sein d'un tableau à deux dimensions (12x8) préalablement rempli de valeurs numériques.

8.3. Dames

Écrivez un algorithme de jeu de dames très simplifié.

L'ordinateur demande à l'utilisateur dans quelle case se trouve son pion (ligne, colonne). On met en place un contrôle de saisie afin de vérifier la validité des valeurs entrées.

Ensuite, on demande à l'utilisateur quel mouvement il veut effectuer: **7** (en haut à gauche), **9** (en haut à droite), **1** (en bas à gauche), **3** (en bas à droite).

Si le mouvement est impossible (on sort du damier), on le signale à l'utilisateur et on s'arrête là. Sinon, on déplace le pion et on affiche le damier résultant, en affichant un **O** pour une case vide et un **X** pour la case où se trouve le pion.

9. Tris

9.1. Tris 1

Écrivez un algorithme qui permette de saisir un nombre quelconque de valeurs et qui les range au fur et à mesure dans un tableau. L'algorithme, une fois la saisie terminée, doit dire si les éléments du tableau sont tous consécutifs ou non. Par exemple, si le tableau est:

12	13	14	15	16	17	18

ses éléments sont consécutifs. En revanche, si le tableau est:

9	10	11	15	16	17	18

ses éléments ne sont pas tous consécutifs.

9.2. Décroissant sélectif

Écrivez un algorithme qui trie un tableau dans l'ordre *décroissant* avec un tri par **sélection**.

9.3. Décroissant bulle

Écrivez un algorithme qui trie un tableau dans l'ordre *décroissant* avec un **tri à bulles**.

9.4. Tri par insertion

Écrivez un algorithme qui trie un tableau dans l'ordre *croissant* avec un **tri par insertion**.

9.5. Inversion

Écrivez un algorithme qui inverse l'ordre des éléments d'un tableau préalablement saisi. **Il est interdit d'utiliser un slice ou une fonction Python existante!**.

9.6. Suppression

Écrivez un algorithme qui permette à l'utilisateur de **supprimer** une valeur d'un tableau préalablement saisi. L'utilisateur donnera d'indice de la valeur qu'il souhaite supprimer. Attention, il ne s'agit pas de mettre la valeur à 0, mais *bel et bien supprimer la case du tableau!*. **Il est interdit d'utiliser la fonction .remove() du tableau**.

9.7. Doublons

Écrivez un algorithme qui permette de saisir les éléments d'un tableau et qui vérifie s'ils sont tous différents. L'algorithme affichera simplement s'il y a **un ou plusieurs doublons** ou **aucun doublon** le cas échéant.

9.8. Fusion

Écrivez un algorithme qui **fusionne** deux tableaux existants dans un troisième, qui sera trié. On présume que les deux tableaux de départs sont préalablement triés.

9.9. Tri shaker

Écrivez un algorithme qui trie un tableau avec le tri **shaker**.

9.10. Tri rapide

Écrivez un algorithme qui trie un tableau avec le tri **rapide**.

10. Fonctions

10.1. Max 1

Écrivez une *fonction* qui demande **deux** paramètres entiers, puis qui *renvoie le nombre maximum*.

10.2. Max 2

Écrivez une *fonction* qui demande **plusieurs** paramètres entiers, puis qui *renvoie le nombre maximum*.

10.3. FizzBuzz

Écrivez une *fonction* nommée **fizzbuzz** prenant un nombre en paramètre. Cette fonction *renverra* :

- **FIZZ** si le nombre *est divisible par 3* ;
- **BUZZ** si le nombre *est divisible par 5* ;
- **FIZZBUZZ** si le nombre est *divisible par 3* ***et*** *par 5* ;
- Dans les autres cas, la fonction renverra *le nombre passé en paramètre*.

10.4. Permis

Écrivez une *fonction* qui vérifie une vitesse.

1. Si la vitesse est inférieure à 70km/h, elle *renverra* ***OK*** ;
2. Sinon, pour chaque tranche de 5km/h au-dessus de la limite, le conducteur perd 1 point sur son permis. La fonction *renverra le nombre de points perdus* ;
3. Au-delà de 12 points perdus, la fonction *renverra* ***Permis suspendu***.

10.5. Séparation de liste

Écrivez une *fonction* qui sépare les **0** et les **1** d'une liste, en mettant les **0** en début et les **1** en fin de liste. Ex: `separate([0,1,0,1,1,1,0]) >>> [0,0,0,1,1,1,1]`.

10.6. Code de somme

Écrivez une *fonction* qui prend en paramètre un nombre **strictement supérieur à 100** et qui permet ensuite de déterminer un code à partir de ce nombre, selon les étapes suivantes :

- Faire la somme des chiffres du nombre passé en paramètre.
- Recommencer l'opération tant que le code est supérieur à 9.

Ex : `sumcode(69810) >>> 669810`

10.7. Chiffres distincts

Écrivez une *fonction* qui prend en paramètre un nombre entier et qui indique par *True* ou *False* si **tous** les chiffres sont différents.

10.8. Nombres palindromes 1

Écrivez une *fonction* qui prend un nombre en paramètre et qui indique par *True* ou *False* si le nombre est palindrome ou non.

Un nombre est un nombre palindrome s'il donne la même valeur en étant lu de gauche à droite et de droite à gauche.

10.9. Nombre palindromes 2

Le plus grand nombre palindrome obtenu comme produit de deux entiers à deux chiffres est $9009 = 91 \times 99$.

Écrivez un *programme* qui permet de trouver le plus grand nombre palindrome obtenu comme produit de deux entiers à trois chiffres.

10.10. Nombres circulaires premiers

Écrivez une *fonction* qui prend en paramètre un nombre entier et qui teste si le nombre est **circulaire premier**. Un nombre est **circulaire premier** si la rotation de ses chiffres sont également des nombres premiers. Ex : 71 est circulaire premier, car 17 est premier et 71 l'est aussi.

Écrivez une sous-fonction *is_prime(number)* pour vous aider.

11. Fonctions récursives

11.1. Somme

Écrivez une fonction **récursive** qui renvoie la somme des nombres entiers jusqu'à un nombre préalablement entré. Exemple avec le nombre 5 : $1 + 2 + 3 + 4 + 5 = 15$

11.2. Syracuse

Écrivez une fonction **récursive** qui affiche les éléments de la *conjecture de Syracuse*. Pour rappel, la conjecture de Syracuse est une suite d'entiers naturels définie de la manière suivante :

- on part d'un nombre entier plus grand que **0** ;
- si ce nombre est **pair**, on le divise par deux ;
- si ce nombre est **impair**, on le multiplie par 3 et on ajoute 1 ;
- on répète l'opération jusqu'à ce que l'on obtienne la valeur **1**.

11.3. Fibonacci

Écrivez une fonction **récursive** qui calcule le n^{ème} élément de la *suite de Fibonacci*. En mathématiques, la *suite de Fibonacci* est une suite de nombres entiers dont **chaque terme successif est la somme des deux termes précédents, et qui commence par *0* et *1***. Par exemple, les dix premiers nombres de la suite sont : $0, 1, 1, 2, 3, 5, 8, 13, 21, 34$.

11.4. Somme digits

Écrivez une fonction **récursive** qui additionne chaque *chiffre* d'un nombre donné. Ex : func(354) = 12.

11.5. Somme harmonique

Écrivez une fonction **récursive** qui affiche le n^{ème} élément de la suite harmonique. Une suite harmonique est la somme des éléments inverses : $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ...$

11.6. Conversion Décimal-Binaire

Écrivez une fonction **récursive** qui convertisse un nombre décimal en un nombre binaire. Ex : func(12) = b1100.

11.7. Calcul du GCD

Écrivez une fonction **récursive** qui calcule le GCD (*Grand Commun Diviseur*) entre deux nombres.

11.8. Tour de Hanoï 1

Écrivez une fonction **récursive** qui affiche les étapes de résolution des Tours de Hanoï.

11.9. Tour de Hanoï 2

Concevez un **programme** qui affiche le déplacement des plateaux (en ascii) pour résoudre le problème des Tours de Hanoï.

11.10. Récursivité mutuelle

Écrivez *deux fonctions* récursives mutuelles permettant de tester si un nombre est pair ou impair. Ex:

```
is_even(5) >>> False
is_odd(7) >>> True
```

12. Fonctions texte

12.1. Compte lettres

Écrivez un algorithme qui demande un mot à l'utilisateur et qui affiche à l'écran le nombre de lettres de ce mot.

12.2. Compte mot

Écrivez un algorithme qui demande une phrase à l'utilisateur et qui affiche à l'écran le nombre de mots de cette phrase. On suppose que les mots ne sont séparés que par des espaces.

12.3. Voyelles

Écrivez un algorithme qui demande une phrase à l'utilisateur et qui affiche le nombre de voyelles contenues dans cette phrase.

12.4. Supprime lettre

Écrivez un algorithme qui demande une phrase à l'utilisateur. Celui-ci entrera ensuite la position d'un caractère à supprimer et la nouvelle phrase doit être affichée.

12.5. Inversion

Écrivez un algorithme qui demande une phrase à l'utilisateur, puis qui inverse l'ordre des mots de la phrase.

12.6. Cryptographie 1

Un des plus anciens systèmes de cryptographie consiste à décaler les lettres d'un message pour le rendre illisible. Ainsi, les **a** deviennent des **b**, etc.

Écrivez un algorithme qui demande une phrase à l'utilisateur et qui la code selon ce principe.

12.7. Cryptographie 2 - le code de César

Une amélioration du principe précédent consiste à opérer un décalage d'un nombre quelconque de lettres.

12.8. Cryptographie 3

Une technique ultérieure de cryptographie consiste à opérer non avec un décalage systématique, mais par une substitution aléatoire. Pour cela, on utilise un *alphabet-clé* dans lequel les lettres se succèdent de façon désordonnée. Par exemple:

C'est cette clé qui va servir ensuite à coder le message.

Écrivez un algorithme qui effectue ce cryptage. L'utilisateur entrera son alphabet-clé (on suppose qu'il le fera sans erreurs), puis une phrase à coder. L'algorithme affichera la phrase codée.

12.9. Cryptographie 4 - le chiffre de Vigenère

Le chiffre de Vigenère est beaucoup plus difficile à briser que les précédents. Il consiste en une combinaison de différents chiffres de César.

On peut en effet écrire 25 alphabets décalés par rapport à l'alphabet normal:
- l'alphabet qui commence par **b** et finit par **yza**;

- l'alphabet qui commence par **c** et finit par **zab**;
- etc.

Le codage va s'effectuer sur le principe du code de César: on remplace la lettre d'origine par la lettre occupant la même place dans l'alphabet décalé. Mais à la différence du chiffre de César, un même message va utiliser non un, mais plusieurs alphabets décalés. Pour savoir quels alphabets seront utilisés, et dans quel ordre, on utilise une clé.

Par exemple, si on prend la clé **VIGENERE**, et la phrase à coder *Il faut coder cette phrase*, nous procéderons comme suit:
- La première lettre (**I**) sera codée en utilisant l'alphabet commençant par **V**. La neuvième lettre de cet alphabet est le **D**.
- La seconde lettre (**l**) sera codée avec l'alphabet commençant par **I**. «L» étant la 12eme lettre de l'alphabet normal, elle deviendra un **t** dans l'alphabet décalé.
- etc.

Écrivez l'algorithme qui effectue un cryptage de Vigenère, en demandant bien sûr au départ la clé à l'utilisateur.

12.10. Pair ou impair

Écrivez un algorithme qui demande un nombre à l'utilisateur. L'ordinateur affiche ensuite si ce nombre est pair ou impair.

12.11. Schpountz

Écrivez différents algorithmes qui génèrent un nombre Schpountz aléatoire tel que:
- $0 =< Schpountz < 2$
- $-1 =< Schpountz < 1$
- $1,35 \leq Schpountz < 1,65$
- Schpountz simule un dé à six faces.
- $-10,5 \leq Schpountz < +6,5$
- Schpountz simule la somme d'un jet de deux dés à 6 faces.

13. Exercices plus complexes

13.1. Le jeu des allumettes

Le jeu des allumettes se joue à deux joueurs. Le principe est simple : 20 allumettes sont alignées. Chaque joueur a le droit de tirer 1, 2 ou 3 allumettes à la fois. Celui qui tire la dernière allumette a perdu.

Programmez le jeu des allumettes. Le jeu doit répondre aux critères suivants :

- Le jeu doit être jouable à deux joueurs humains. Il demandera le nom des deux joueurs.
- Le jeu choisira au hasard un des deux joueurs.
- Le jeu doit afficher le nombre d'allumettes restantes (au début, 20) :

IIIIIIIIIIIIIIIIIII

- Le jeu doit vérifier que le joueur ne tire qu'une à trois allumettes.
- Le jeu s'arrête quand un joueur a tiré la dernière allumette. Il affichera un message indiquant quel joueur a perdu.

13.2. Le jeu des allumettes 2

Modifiez le jeu des allumettes pour lui ajouter les fonctionnalités suivantes :

- Le jeu peut maintenant être joué seul contre l'ordinateur ou à deux joueurs. Le joueur indiquera le mode de jeu (seul/ deux joueurs).
- En cas de jeu contre l'ordinateur, programmez une intelligence artificielle qui choisira au hasard une à trois allumettes lors de son tour de jeu.

13.3. Le jeu des allumettes 3

Modifiez l'intelligence artificielle de telle sorte qu'elle ne tire plus des allumettes au hasard, mais tente d'empêcher le joueur de gagner. On pourra aussi jouer avec un nombre quelconque d'allumettes (entre 20 et 100).

13.4. Le jeu du morpion

Le jeu du morpion (ou TicTacToe) est un jeu à deux joueurs, dont le principe est d'aligner sur une grille de 9 cases 3 symboles identiques. Programmez un morpion avec les contraintes suivantes :

- Le jeu se joue à deux joueurs. Le programme demande le nom des deux joueurs, puis choisira au hasard qui va commencer. Le joueur 1 jouera avec les « X » et le joueur 2 avec les « O ».
- Le jeu affichera le plateau de jeu avec des chiffres de 1 à 9, indiquant la position des cases vides, ou un symbole « X » ou « O » en cas de case jouée.
- Le jeu vérifiera que le joueur joue dans une case vide.
- Si un joueur aligne ses trois symboles, le jeu indiquera qu'il a gagné.
- Si le plateau est complètement rempli, il déclarera une égalité entre les joueurs.
- Le programme s'arrête dès la fin du jeu.

13.5. Le jeu du morpion 2

Modifiez le jeu du morpion pour le rendre jouable à deux joueurs ou un joueur contre l'ordinateur. L'ordinateur jouera systématiquement les « O ». Programmez une intelligence artificielle qui jouera dans une case libre au hasard.

13.6. Le jeu du morpion 3

Modifiez le jeu du morpion précédent en modifiant l'IA de telle sorte qu'elle « réfléchisse où elle va jouer, selon les contraintes suivantes :

- Si elle peut gagner au prochain coup, alors elle joue dans la case qui la fera gagner.
- Si elle peut gagner en jouant dans plusieurs cases, elle choisira au hasard une des deux cases qui la fera gagner.
- Si elle ne peut pas gagner, elle va vérifier si le joueur peut gagner au prochain coup. Si c'est le cas, elle va jouer dans la case qui va empêcher le joueur de gagner.
- Si elle ne peut pas gagner, et que le joueur ne peut pas gagner à son prochain tour, elle jouera au hasard dans les cases restantes.

13.7. Pendu 1

Écrivez un jeu de pendu. Le jeu contient un fichier dictionnaire. Le programme charge le fichier dans une liste, puis sélectionne au hasard un mot à faire découvrir au joueur. Gérez au moyen de différentes fonctions l'affichage du mot à découvrir, l'affichage du pendu et la gestion des lettres entrées.

13.8. Pendu 2

Modifiez le programme du pendu pour faire en sorte que le joueur puisse indiquer au programme une taille de mot. Le programme sélectionnera un mot de la taille donnée au hasard. Lors du chargement du fichier dictionnaire, classez les mots en fonction de leur taille dans un dictionnaire.

13.9. Pendu 3

Ajoutez au programme une gestion des scores. Le programme doit stocker les 10 meilleurs joueurs. Le meilleur joueur est celui qui trouve le mot le plus long possible en jouant le moins de coups possibles. Stockez les informations dans un fichier. Les informations à stocker sont : le nom du joueur, le mot trouvé, la taille du mot et le nombre de coups joués. Affichez le leaderboard en début de partie, et en fin de partie si le joueur entre dans le leaderboard.

13.10. 2048 1

Écrivez le jeu du **2048** en version console. Le but du jeu est d'afficher un tableau de **4x4** cases avec des tuiles de couleur. Chaque tuile possède une valeur numérique entre 2 et 2048 (toujours une puissance de deux). En appuyant sur une des flèches du clavier, on va **pousser** toutes les tuiles dans la direction indiquée. Si deux tuiles identiques se collisionnent lors de la compression, elles **fusionnent** en une tuile de rang supérieur. Après la compression, **deux** nouvelles tuiles apparaissent. Ces tuiles portent aléatoirement les numéros **2** ou **4**.

Le but est d'atteindre la tuile 2048.

Analysez le problème pour en déduire les fonctions à écrire.

13.11. 2048 2

Écrivez le jeu du **2048** avec **tkinter**, avec une *frame*, des *labels* et des *fonctions*.

13.12. SameGame 1

Concevez le jeu du **SameGame** en version console. Le « SameGame » est un jeu de réflexion dont le but est d'enlever toutes les tuiles du plateau en cliquant sur des groupes de tuiles de la même couleur. Vous ne pouvez retirer que les tuiles qui sont connectées verticalement ou horizontalement. Plus vous enlevez de tuiles en un seul clic, plus vous obtenez de points. Le jeu se termine lorsqu'il n'est plus possible de faire de mouvement.

Vous pouvez utiliser les chiffres (ou des lettres) à la place des couleurs, ou vous amuser avec la bibliothèque **colorama**.

13.13. SameGame 2

Ajoutez une interface graphique Tkinter au jeu.

13.14. Démineur 1

Concevez le jeu du **démineur** en version console. Le but du jeu est de détecter des mines placées aléatoirement dans un champ de mines en y posant un *drapeau*. Concevez ce projet en **TDD** avec *Pytest*. Vous pouvez utiliser des bibliothèques tierce (comme *tabulate* et *colorama*) pour l'interface graphique.

Faites en sorte de ne pas tomber sur une mine dès la première case.

13.15. Démineur 2

Ajoutez une interface graphique Tkinter au jeu.