Voici quelques consignes pour les séances de TP :

- Vous pouvez utiliser vos ordinateurs personnels ou bien ceux du lycée.
- Pour lancer Spyder au lycée :
  - Ouvrir le dossier `Informatique` qui se trouve sur le bureau.
  - Ouvrir au choix `Spyder (Jupiter)` ou bien `Spyder 2`. Si l'un de ces deux programmes ne fonctionne pas, utilisez l'autre.
  - Dans l'éditeur de texte, supprimez la ligne `exit()` si elle y apparaît.
- Au début de chaque TP, pensez à enregistrer votre fichier source (au format `.py`) sur une clé USB ou sur le disque `U:`. Si vous voulez enregistrer des documents sur les ordinateurs du lycée, utilisez toujours le disque `U:`.
- Vos fonctions doivent être testées avec les exemples donnés dans l'énoncé et tous ces tests doivent apparaître dans votre fichier source. Vous pouvez ensuite mettre un `#` devant les tests pour éviter qu'ils ne s'exécutent à chaque fois.
- La correction sera disponible après le TP sur la page du cours :

`https://informatique-lhp.fr`

Les exercices 1, 2 et 3 sont facultatifs. Cela signifie que vous pouvez ne pas faire une question si la solution vous paraît évidente **(et seulement si elle vous paraît vraiment évidente)**.

Exercice 1. Calcul de la moyenne

Soient n_1, n_2, n_3 des nombres réels. La moyenne arithmétique de n_1, n_2, n_3 est le réel :

$$\frac{n_1 + n_2 + n_3}{3}$$

1. Écrire une fonction `moy` qui prend en entrée trois entiers `n1`, `n2`, `n3` et renvoie leur moyenne arithmétique. Vérifier que :

`moy(5, 6, 7)` vaut `6.0` `moy(1, 1, 1)` vaut `1.0` `moy(-4, 6, 8)` vaut `3.333...`

Soient $n_1, n_2, n_3, w_1, w_2, w_3$ des nombres réels tels que $w_1 + w_2 + w_3 \neq 0$. La moyenne arithmétique de n_1, n_2, n_3 pondérée par w_1, w_2, w_3 est le réel :

$$\frac{w_1 n_1 + w_2 n_2 + w_3 n_3}{w_1 + w_2 + w_3}$$

2. Écrire une fonction `moy_ponderee` qui prend en entrée trois entiers `n1`, `n2`, `n3`, trois poids `w1`, `w2`, `w3` et renvoie la moyenne arithmétique de `n1`, `n2`, `n3` pondérée par les coefficients `w1`, `w2`, `w3`. Vous devez vérifier que la somme des poids est non nulle à l'aide d'un `assert`. Par exemple :

`moy_ponderee(5, 6, 7, 1, 1, 1)` vaut `6.0`
`moy_ponderee(5, 3, 7, 6, -7, 2)` vaut `23.0`
`moy_ponderee(1, 2, 3, 4, 5, 6)` vaut `2.1333...`
`moy_ponderee(5, 6, 7, 1, 1, -2)` déclenche une erreur

Soient $n_1, n_2, n_3, w_1, w_2, w_3$ des nombres réels tels que $w_1 + w_2 + w_3 \neq 0$. La moyenne géométrique de n_1, n_2, n_3 pondérée par w_1, w_2, w_3 est le réel :

$$(n_1^{w_1} n_2^{w_2} n_3^{w_3})^{1/(w_1+w_2+w_3)}.$$

3. Écrire une fonction `moy_geom_ponderee` qui prend en entrée trois entiers ainsi que trois poids, et calcule la moyenne géométrique associée. Vous devez vérifier que la somme des poids est non nulle à l'aide d'un `assert`. Par exemple :

   `moy_geom_ponderee(5, 6, 7, 1, 1, 1)` vaut `5.943...`
   `moy_geom_ponderee(5, 3, 7, 6, -7, 2)` vaut `350.080...`
   `moy_geom_ponderee(1, 2, 3, 4, 5, 6)` vaut `1.955...`
   `moy_geom_ponderee(5, 6, 7, 1, 1, -2)` déclenche une erreur

Exercice 2. Coefficients binomiaux

La factorielle d'un entier $n \in \mathbb{N}$ est définie par :

$$\begin{cases} n! = 1 & \text{si } n = 0. \\ n! = n(n-1)(n-2)\dots 2 \cdot 1 & \text{si } n > 0. \end{cases}$$

Pour $n, k \in \mathbb{N}$, le coefficient binomial $\binom{n}{k}$ (se lit "k parmi n") est défini par :

$$\begin{cases} \binom{n}{k} = \dfrac{n!}{k!(n-k)!} & \text{si } k \leqslant n \\ \binom{n}{k} = 0 & \text{si } k > n \end{cases}$$

1. Écrire une fonction `fact` qui prend en entrée un entier n et renvoie $n!$. Votre fonction déclenchera une erreur si l'argument est strictement négatif. Par exemple :

`n`	0	1	2	5	12	-1
`fact(n)`	1	1	2	120	479001600	Erreur

2. Écrire une fonction `binom` qui prend en entrée deux entiers `n, k` et renvoie $\binom{\texttt{n}}{\texttt{k}}$. Votre fonction doit renvoyer un `int` (pas un `float`). De plus, votre fonction déclenchera une erreur si $k < 0$ ou $n < 0$. Par exemple :

`n`	7	7	8	5	-1	5
`k`	0	7	4	10	5	-2
`binom(n,k)`	1 (et pas `1.0`)	1 (et pas `1.0`)	70 (et pas `70.0`)	0	Erreur	Erreur

Exercice 3. Conversion de durées

1. Écrire une fonction `hms_to_s` qui prend en entrée trois entiers positifs :
   - `h` un nombre d'heures,
   - `m` un nombre de minutes,
   - `s` un nombre de secondes,

   et renvoie le nombre de secondes correspondantes. Par exemple :

   `hms_to_s(0, 0, 50)` vaut `50`
   `hms_to_s(0, 1, 0)` vaut `60`
   `hms_to_s(1, 0, 0)` vaut `3600`
   `hms_to_s(2, 1, 50)` vaut `7310`
   `hms_to_s(22, 100, 50)` vaut `85250`

2. Expliquer ce que fait la commande suivant :

```
print("On a les egalites 1+1 =", 1+1, "et 2*3 =", 2*3)
```

3. Écrire une fonction `print_duree` qui prend en entrée un nombre de secondes s et affiche cette durée sous le format `heures - minutes - secondes`. Votre fonction n'affichera rien si s est strictement négatif. Si vous n'y arrivez pas, vous pouvez suivre les indications ci-dessous. Par exemple :

   `print_duree(10)` affiche `0 - 0 - 10` `print_duree(60)` affiche `0 - 1 - 0`
   `print_duree(3600)` affiche `1 - 0 - 0` `print_duree(33333)` affiche `9 - 15 - 33`
   `print_duree(86400)` affiche `24 - 0 - 0`
   `print_duree(-5)` n'affiche rien (même pas `None`)

Indications (essayez de résoudre l'exercice sans lire ce qui suit). On définit successivement les entiers suivants :

- h le nombre d'heures contenues dans s secondes (h est un entier).
- s_1 le nombre de secondes qu'il reste lorsqu'on enlève h heures à s secondes .
- m le nombre de minutes contenues dans s_1 secondes (m est un entier).
- s_2 le nombre de secondes qu'il reste lorsqu'on enlève m minutes à s_1 secondes.

Une durée de s secondes correspondent alors à h heures, m minutes et s_2 secondes.

4. (a) Expliquer comment calculer h, s_1, m et s_2 en utilisant des divisions entières (`//`) et des modulos (`%`).
   (b) Répondre à la question 3.

Exercice 4. Nombre de diviseurs d'un entier

Soit D l'entier maximal tel qu'il existe un entier entre 1 et 15 000 avec D diviseurs.

1. À l'aide de Python, calculer D (réponse : 72).
2. Afficher tous les entiers compris entre 1 et 15 000 ayant D diviseurs (réponse : `10080, 12600, 13860`). Votre programme ne doit pas afficher `None`.

Exercice 5. Morpion

Dans cet exercice, nous allons utiliser la bibliothèque "Tkinter" qui permet de créer des interfaces graphiques avec Python. Récupérez le fichier annexe disponible à l'adresse :

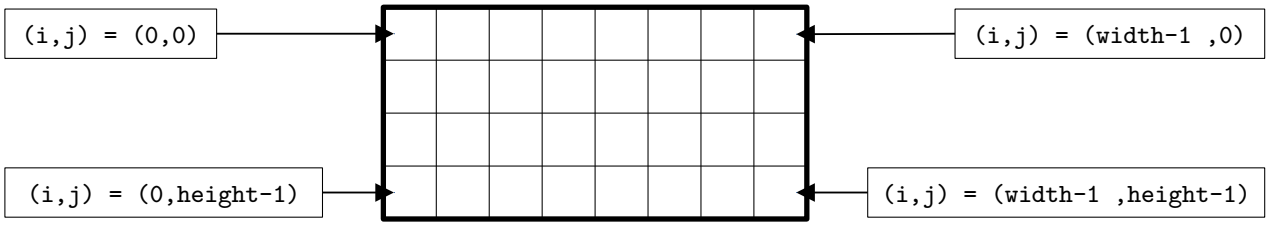
`https://informatique-lhp.fr`

Lorsque vous exécutez ce fichier, une nouvelle fenêtre doit s'ouvrir. Chaque clic dans cette fenêtre créé une ligne entre le coin supérieur gauche et le pointeur de votre souris. Si vous ne parvenez pas à exécuter le fichier, il faut installer Tkinter sur votre PC. Pour cela, tapez la commande suivante dans la console de Spyder (nécessite une connexion internet) :

```
pip install tk
```

Si ça ne fonctionne toujours pas, utilisez un ordinateur du lycée.

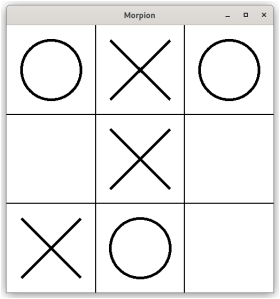
La fonction `open_window` présente dans le fichier permet d'ouvrir une nouvelle fenêtre graphique. Elle renvoie un ***canevas***, c'est à dire une zone de la fenêtre où on va pouvoir afficher des dessins. Un canevas est composé de pixels répartis sur une grille composée de `width` colonnes et `height` lignes. Chaque pixel est repéré par un couple `(i, j)` avec $0 \leqslant$ `i` $\leqslant$ `width - 1` et $0 \leqslant$ `j` $\leqslant$ `height - 1` où `i` est le numéro de la colonne et `j` est le numéro de la ligne. La colonne de gauche correspond à `i = 0` et celle de droite à `i = width - 1`. La ligne du haut correspond à `j = 0`, et celle du bas à `j = height - 1`.



Voici les commandes dont vous aurez besoin pour cet exercice :

- `can = open_window(width, height, title)` ouvre une fenêtre graphique où :
  - `width` de type `int` est le nombre de pixels dans la largeur du canevas.
  - `height` de type `int` est le nombre de pixels dans la hauteur du canevas.
  - `title` de type `str` est le titre de la fenêtre.
- `can.create_line(p1, p2, width = 2)` trace une ligne entre les pixels de coordonnées `p1` et `p2`. Les paramètres `p1` et `p2` sont des couples d'entiers. L'argument optionnel `width` permet d'indiquer l'épaisseur de la ligne.
- `can.create_oval(p1, p2)` trace un ovale inscrit dans le rectangle dont `p1` est le pixel en haut à gauche et `p2` est le pixel en bas à droite.
- `can.mainloop()` lance une boucle qui attend que l'on ferme la fenêtre.
- `can.bind('<Button-1>', on_click)` exécute la fonction `on_click` (définie dans le fichier) à chaque fois que l'utilisateur clique dans la fenêtre avec la souris. Dans la fonction `on_click`, les deux entiers `event.x` et `event.y` correspondent aux coordonnées du pixel ayant été cliqué.
- `tk.messagebox.showinfo(titre, msg)` affiche un message dans une nouvelle fenêtre. Les chaînes de caractères `titre` et `msg` contiennent le titre de la fenêtre et le message.

Le but de l'exercice est de créer une interface graphique pour jouer au morpion. Vous devez ouvrir une fenêtre graphique pour y tracer une grille 3×3. À chaque clic dans la grille, la case correspondante doit se remplir avec une croix ou un cercle (en fonction du joueur qui est en train de jouer). À la fin de la partie, affichez le résultat (victoire de l'un des joueurs ou égalité) grâce à la commande `tk.messagebox.showinfo`.



Exercices à rendre au plus tard le 17/09/2023 à 20h

Voici les règles pour les rendus :

- Toutes les semaines, vous aurez des exercices à faire.
- Envoyez vos exercices à l'adresse `rendu.itc.mpsi1@gmx.fr` pour les MPSI 1 et `rendu.itc.mpsi2@gmx.fr` pour les MPSI 2. N'utilisez pas une autre adresse mail pour m'envoyer vos rendus. En revanche, si vous avez une question, ne l'envoyez pas sur `rendu.itc.mpsiX@gmx.fr` mais sur `antoine.grospellier@gmx.fr`.
- Pour me rendre vos exercices, vous devez m'envoyer un fichier source (avec l'extension .py) en pièce jointe de votre mail. Pas de copier/coller du code dans le message du mail, pas de photo prise par téléphone, pas de scan, pas de rendu papier ... Seulement le fichier .py en pièce jointe du mail.
  Si votre boite mail refuse d'envoyer le fichier `.py` (c'est parfois le cas de Gmail), changez l'extension du fichier en `.txt` et envoyez le moi avec cette extension.
- Vous avez tout à fait le droit de travailler à plusieurs ou de vous faire aider pour les rendus de TP. En revanche, au moment où vous écrivez le programme, vous devez commencer d'un fichier vierge et écrire le programme par vous-même. Si deux personnes m'envoient le même fichier, elles seront sanctionnées.
- Les tests donnés dans l'énoncé doivent apparaître dans votre fichier source et votre fonction doit marcher sur ces exemples. Une fois que vous avez vérifié que les tests fonctionnent, vous pouvez les mettre en commentaire avec des `#`.

- Si une question de l'exercice demande une réponse en français, vous l'écrivez directement dans le fichier source sous la forme d'un commentaire.
- Certains rendus seront corrigés de manière automatique, cela signifie que :
  - Il ne doit pas y avoir d'erreur lors de l'exécution du fichier.
  - Les noms de fonctions doivent être ceux demandés par l'énoncé (par exemple, respectez les majuscules s'il y en a).
- Les retards seront sanctionnés. Un fichier envoyé plus de 12h après la date limite est considéré comme non rendu. Les excuses du style : "Je suis à l'internat et la connexion est mauvaise" ou "Le mail est resté bloqué dans la boite d'envoi" ne sont pas valables.

Exercice 6. Évaluation d'expressions

Trouver (sans utiliser Python) la valeur et le type des expressions suivantes :

`56//10` `2**5 % 10` `12/4`

Vérifiez vos résultats à l'aide de Python et de la commande `type`.

Exercice 7. Prix toutes taxes comprises (TTC)

1. Écrire une fonction `prix_TTC` qui prend en entrée un prix hors taxes ainsi qu'un taux de TVA exprimé en pourcentage, et renvoie le prix TTC correspondant. Par exemple :

   `prix_TTC(123, 10)` vaut `135.3`, `prix_TTC(321, 5.5)` vaut `338.655`.

2. Écrire une fonction `prix_HT` qui prend en entrée un prix TTC ainsi qu'un taux de TVA exprimé en pourcentage, et renvoie le prix hors taxes correspondant. Par exemple :

   `prix_HT(135.3, 10)` vaut `123.0`, `prix_HT(338.655, 5.5)` vaut `321.0`.

Exercice 8. Suite numérique

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par :

$$\begin{cases} u_0 = 0 \\ u_{n+1} = -2u_n + 1 \qquad \text{pour tout } n \geqslant 0 \end{cases}$$

Écrire une fonction `get_u` qui prend en entrée un entier n et renvoie u_n. Par exemple :

`n`	0	1	2	15
`get_u(n)`	0	1	-1	10923

Exercice 9. Zones de couleurs (exercice facultatif)

Ceci est un exercice du site France-IOI où vous pouvez apprendre différents langages de programmation, dont le langage Python :

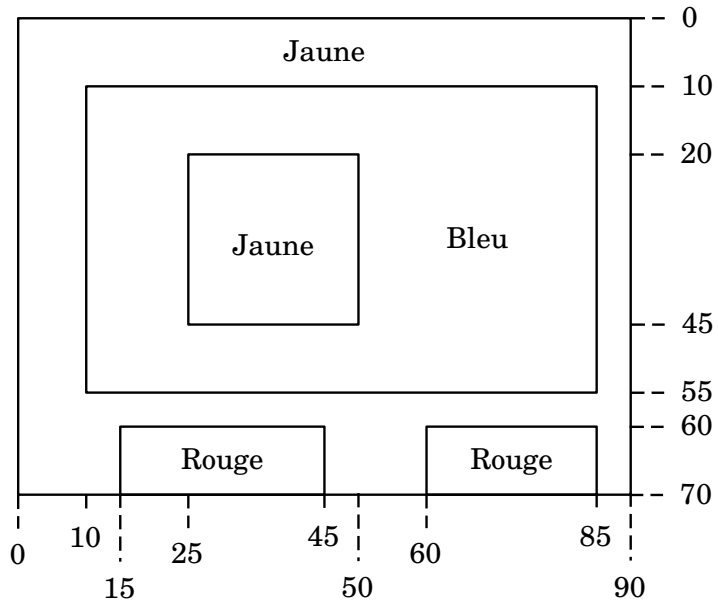
`http://www.france-ioi.org/algo/task.php?idChapter=648&idTask=496`.

Ce que doit faire votre programme. Sur une table est placée une feuille de papier rectangulaire de 90 cm de large et 70 cm de haut, composée de zones de différentes couleurs, comme le décrit la figure ci-dessous. Une personne place un jeton où elle le souhaite sur la table, à l'exception des frontières entre les différentes zones.

On vous donne en entrée les coordonnées du jeton sur la feuille par rapport à l'origine en haut à gauche, sous la forme d'une abscisse et d'une ordonnée. Votre programme devra **afficher** l'un des textes suivants, en fonction de la couleur sur laquelle se trouve le jeton :

- "En dehors de la feuille"
- "Dans une zone jaune"
- "Dans une zone bleue"
- "Dans une zone rouge"

Essayez d'écrire votre programme de sorte qu'il y ait au maximum une condition par possibilité de texte affiché. Appelez votre fonction « `zone` ».



Exemples :

`zone(16, 12)` affiche `"Dans une zone bleue"`,
`zone(30, 22)` affiche `"Dans une zone jaune"`,
`zone(64, 62)` affiche `"Dans une zone rouge"`,
`zone(-5, 86)` affiche `"En dehors de la feuille"`.

Exercices à rendre au plus tard le 24/09/2023 à 20h

Exercice 10. Distance SNCF

On se place dans le plan $\mathbb{R}^2$ et on note $\| \ \|_2$ la distance Euclidienne :

$$\|(x, y)\|_2 = \sqrt{x^2 + y^2} \qquad \text{pour tout } (x, y) \in \mathbb{R}^2.$$

La distance SNCF $d(u, v)$ de deux vecteurs $u \in \mathbb{R}^2$ et $v \in \mathbb{R}^2$ est définie par :

$$d(u, v) = \begin{cases} \|u - v\|_2 & \text{si } u \text{ et } v \text{ sont colinéaires,} \\ \|u\|_2 + \|v\|_2 & \text{sinon.} \end{cases}$$

On rappelle que deux vecteurs $u = (x_1, y_1) \in \mathbb{R}^2$ et $v = (x_2, y_2) \in \mathbb{R}^2$ sont colinéaires si et seulement si $x_1y_2 - y_1x_2 = 0$. Le nom "distance SNCF" vient du fait que, sauf si la ville de départ et la ville de destination sont alignées avec Paris, il est souvent obligatoire de passer par Paris pour voyager avec la SNCF (la ville de Paris est ici placée à l'origine).

On souhaite implémenter la distance SNCF en Python. Pour cela, on représente un point du plan par un couple de flottants `(x,y)`. Pour éviter les problèmes de précision, on considérera que deux points `(x1,y1)` et `(x2,y2)` sont colinéaires lorsque $|\texttt{x1} \times \texttt{y2} - \texttt{y1} \times \texttt{x2}| \leqslant 10^{-10}$.

Écrire une fonction `dist_SNCF` qui renvoie la distance SNCF entre les deux points passés en paramètre. Cette fonction doit prendre en entrée deux couples de flottants et non quatre flottants. Par exemple :

`dist_SNCF((2,1),(1,1))` vaut `3.6502...`

`dist_SNCF((2,1),(4,2))` vaut `2.2360...`

Exercice 11. Visite au musée

Un musée propose plusieurs types d'entrées à différents tarifs :

- Tarif plein : 11€.
- Entrée après 16h : 6€70
- Tarif pour les étudiants : 7€90
- Tarif pour les moins de 10 ans : 3€
- Tarif pour les plus de 60 ans du lundi au vendredi : 4€90
- Tarif pour les plus de 60 ans le samedi et le dimanche : 9€

Écrire une fonction `meilleur_tarif` qui renvoie le tarif le plus intéressant pour un visiteur. Votre programme aura quatre paramètres d'entrée :

- Un paramètre de type `int` qui représente l'âge du visiteur.
- Un paramètre de type `float` qui représente l'heure d'arrivée.
- Un paramètre de type `bool` qui indique si le visiteur est un étudiant.
- Un paramètre de type `str` qui donne le jour de visite (`"lundi"`, `"mardi"`, `"mercredi"`, `"jeudi"`, `"vendredi"`, `"samedi"` ou `"dimanche"`).

Vérifier que :

`meilleur_tarif(9, 8., False, "lundi")` vaut `3`,
`meilleur_tarif(22, 17.5, True, "dimanche")` vaut `6.7`,
`meilleur_tarif(30, 12., True, "samedi")` vaut `7.9`,
`meilleur_tarif(65, 10.5, False, "lundi")` vaut `4.9`,
`meilleur_tarif(65, 10., False, "samedi")` vaut `9`,
`meilleur_tarif(50, 15.5, False, "lundi")` vaut `11`.

Exercice 12. Fléchettes (exercice facultatif)

Ceci est un exercice du site France-IOI :

`http://www.france-ioi.org/algo/task.php?idChapter=656&idTask=2209`.

Vous rencontrez un groupe d'amateurs de fléchettes. Ces joueurs sont de grands passionnés et aiment jouer sur des cibles de tailles variées. Cependant, leurs cibles se font vieilles et mériteraient bien d'être changées ! Vous profitez donc de votre passage parmi eux pour leur imprimer de nouvelles cibles.

Les cibles à imprimer sont de la forme suivante (avec 4 lettres à gauche et 5 lettres à droite) :

```
                    aaaaaaaaa
aaaaaaa             abbbbbbba
abbbbba             abcccccba
abcccba             abcdddcba
abcdcba             abcdedcba
abcccba             abcdddcba
abbbbba             abcccccba
aaaaaaa             abbbbbbba
                    aaaaaaaaa
```

Ce que doit faire votre fonction. Votre fonction nommée `afficher_cible` doit prendre en entrée un unique entier : le nombre de lettres `nbLettres` ($1 \leqslant$ `nbLettres` $\leqslant 26$) à utiliser. Elle doit ensuite afficher la cible correspondante (comme indiqué sur la figure ci-dessus). Votre fonction doit vérifier que $1 \leqslant$ `nbLettres` $\leqslant 26$.

Indication. On pourra utiliser la fonction `min` qui renvoie le minimum de ses arguments et s'inspirer du programme suivant :

```
lettres = "abcdefghijklmnopqrstuvwxyz"
print(lettres[0])          # Affiche la lettre a avec retour à la ligne
print(lettres[0], end='')  # Affiche la lettre a sans retour à la ligne
print(lettres[4], end='')  # Affiche la lettre e sans retour à la ligne
print(lettres[10], end='') # Affiche la lettre k sans retour à la ligne
print()                    # Affiche un retour à la ligne
print(lettres[25])         # Affiche la lettre z avec retour à la ligne
```