Afficher un patron de chiffres ¶

Définissez une **fonction** nommée `chiffres` qui accepte en entrée **deux** nombres entiers m et n, et qui affiche m lignes de n colonnes en utilisant le patron suivant de chiffres. La 1re ligne débute par 0, la seconde par 1, et ainsi de suite jusqu'à la 10e qui débute par 9. Ensuite, le compteur retourne à 0 pour la 11e ligne, puis à 1 pour la 12e, ***etc***. Pour les colonnes, les chiffres sont toujours incrémentés de 1 par rapport à la colonne précédente, sauf pour le 9 qui passe à 0.

Par exemple, l'appel `chiffres(4, 7)` doit produire:

```
0123456
1234567
2345678
3456789
```

et l'appel `chiffres(12, 23)` doit produire:

```
01234567890123456789012
12345678901234567890123
23456789012345678901234
34567890123456789012345
45678901234567890123456
56789012345678901234567
67890123456789012345678
78901234567890123456789
89012345678901234567890
90123456789012345678901
01234567890123456789012
12345678901234567890123
```

Notez bien que votre fonction ne doit **rien** retourner, seulement **afficher** à la console.

Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
def chiffres(m, n):
    for i in range(m):
        ligne = str(i % 10)
        for j in range(1, n):
            ligne += str((i+j) % 10)
        print(ligne)
```

Calculer l'énergie d'un bâtiment

Une façon **d'approximer** l'énergie consommée par un bâtiment est de considérer les facteurs suivants:

1. la surface du bâtiment (en pi^2)
2. la densité d'énergie requise pour le climat local (en BTU/pi^2)
3. la qualité de l'isolation (valeur R)

La densité d'énergie requise pour le climat québécois est typiquement entre 40 et 50 BTU par pied carré de plancher. La qualité de l'isolation se quantifie selon le facteur $R > 1$. Plus le facteur est élevé, plus l'isolation est bonne.

Pour cet exercice, on vous demande d'écrire une fonction nommée `energie` qui estime l'énergie nécessaire pour chauffer un bâtiment de surface S, pour une densité d'énergie d, et un niveau d'isolation R. Votre fonction doit calculer l'énergie $E = \frac{S \times d}{R}$ en acceptant les trois arguments positionnels suivants:

1. la densité d d'énergie;
2. la valeur R de l'isolation;
3. une **liste** de couples (longueur et largeur) spécifiant les dimensions des pièces du bâtiment.

Par exemple, l'appel `energie(50, 16, [(15, 20), (10, 10), (12, 8)])` doit produire `1550.0` BTU.

Bravo!

Votre score est **100/100**.

Il vous reste **3 soumissions** avec rétroaction.

Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
def energie(d, R, pièces):
    # calculer la surface totale des pièces
    surface = 0
    for x, y in pièces:
        surface += x*y
    # calculer l'énergie du bâtiment
    return surface*d/R
```

Trouver l'objet le plus proche

Définissez une fonction nommée `plus_proche` qui accepte **deux** arguments:

1. un **couple** (x, y) qui définit une position dans le plan cartésien;
2. et un **dictionnaire** qui contient des noms d'objet (chaîne de caractères) associés à leurs positions dans le même plan;

et qui retourne le **nom** de l'objet qui est le **plus proche** de la position spécifiée. Dans le contexte de cet exercice, pour mesurer la distance entre deux positions $p_1 = (x, y)$ et $p_2 = (a, b)$, on vous fournit la fonction `distance` qui retourne le résultat de l'expression $\sqrt{(x-a)^2 + (y-b)^2}$. N'hésitez pas à appeler la fonction `distance` à partir de la vôtre.

Par exemple, l'appel:

```
plus_proche((2, 5), {'obj1': (1, 3), 'obj2': (3, 4), 'obj3': (6, 5)})
```

doit retourner `'obj2'`. Dans le cas particulier où **plusieurs** objets sont à **égale** distance de la position spécifiée, votre fonction doit retourner celui dont le nom vient en **premier** dans l'ordre lexicographique (utilisez `<` entre les chaînes à comparer). Dans le cas d'un dictionnaire **vide**, votre fonction doit retourner `None`.

Indice: on peut utiliser l'expression `float('inf')` pour construire un nombre qui correspond à **l'infini**. Ce nombre possède la particularité d'être plus grand que tous les autres nombres à virgule flottante. On peut donc s'en servir pour initialiser une distance minimum qui sera plus grande que n'importe quelle autre distance que l'on pourrait calculer par la suite.

Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
def plus_proche(position, objets):
    dist_min = float('inf')
    réponse = None
    for nom, pt in objets.items():
        dist = distance(position, pt)
        if dist < dist_min:
            dist_min = dist
            réponse = nom
        elif dist == dist_min and nom < réponse:
            réponse = nom
    return réponse
```

Déterminer les diviseurs communs

Définissez une **fonction** nommée `diviseurs` qui accepte en argument un nombre **arbitraire** de nombres entiers, et returne la **liste** de tous leurs diviseurs communs. Un nombre x tel que $1 < x < y$ est dit diviseur de y si et seulement si le **reste** de la division **entière** $\frac{y}{x}$ est égale à zéro.

Par exemple, l'appel `diviseurs(24, 12, 36)` doit produire la liste `[2, 3, 4, 6]`, car tous les nombres de cette liste divisent 24, 12 et 36. Notez que dans **notre** définition des diviseurs de n, les valeurs 1 et n sont **excluses**. Dans le cas où **aucun** diviseur commun n'existe, votre fonction doit retourner une liste vide.

Rappel: le reste de la division entière s'obtient avec l'opérateur `%` (modulo), et l'argument **étoilé** est introduit à la leçon #14.

Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
def diviseurs(*nombres):
    réponse = []
    # pour tous les nombres inférieurs au plus petit des arguments
    for i in range(2, min(nombres)//2+1):
        # vérifier si le nombre divise tous les arguments
        for nb in nombres:
            if nb % i != 0:
                # échec de la vérification
                break
        else:
            # succès de la vérification
            réponse.append(i)
    # retourner les diviseurs trouvés
    return réponse
```

Construire la chaîne des arguments nommés

Définissez une fonction nommée `signature` qui accepte un nombre **arbitraire** d'arguments nommés, et qui **retourne** la signature de ces arguments, c'est-à-dire la chaîne de caractère qui énumère les noms et les valeurs de ces arguments, séparés par des virgules.

Par exemple, si vous faites l'appel `signature(a=1, b=2, c=3)`, la fonction doit retourner la chaîne `'a=1, b=2, c=3'`.

Indices:

2. utilisez l'argument **doublement étoilé** pour les arguments nommés (voir leçon #14);
3. utilisez la fonction join pour joindre des chaînes de caractères.

Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
def signature(**kargs):
    args = []
    for k, v in kargs.items():
        args.append(f'{k}={v}')
    return ', '.join(args)
```