Fonction

1 Fonction

Fonction

Les fonctions comme valeurs

Exercice 1 : Dérivée numérique

On se donne une fonction numérique f dont on souhaite obtenir une approximation de la dérivée $f'(x)$. Pour cela, on utilise les deux approximations suivantes

$$f'(x) \approx \frac{f(x+\varepsilon) - f(x)}{\varepsilon} \quad \text{et} \quad f'(x) \approx \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon}.$$

1. Écrire les fonctions `derive_1(f: function, x: float, eps: float) -> float` et `derive_2`, de même signature, qui prennent en entrée f, x et ε et qui renvoient respectivement l'approximation de $f'(x)$ donnée par la première et la seconde formule.
2. Testez la fonction `deriv_1` avec $f(x) \coloneqq x^2$ puis $f(x) \coloneqq \ln x$ pour $x = 1$ et différentes valeurs de ε de la forme 10^{-n}. Pour quelle valeur de n l'approximation est-elle la meilleure ?
3. Répétez l'expérience avec la fonction `derive_2`. Quelle conclusion pouvez-vous en tirer ?

Sortie anticipée

Exercice 2 : Doublon

Écrire une fonction `doublon(a)` prenant en entrée une liste `a` et renvoyant `True` si `a` possède un doublon et `False` sinon. Par exemple, `doublon([3, 4, 7, 3, 2])` devra renvoyer `True` car 3 est présent deux fois dans la liste. Notre fonction devra sortir dès qu'un doublon est trouvé.

Exercice 3 : Mêmes éléments

Écrire une fonction `memes_elements(u: list[int], v: list[int]) -> bool` déterminant si les listes u et v possèdent les mêmes éléments, peu importe l'ordre et leur nombre d'occurences. Par exemple

```
memes_elements([7, 3, 5, 3], [3, 7, 5])
```

devra répondre `True` et `memes_elements([1, 7, 2], [2, 1])` devra répondre `False` car 7 est présent dans la première liste mais pas dans la seconde.

Exercice 4 : Chaine bien parenthésée

On dit qu'une chaine de caractères s constituée de `'('` et de `')'` est bien parenthésée lorsque chaque parenthèse ouvrante est correctement fermée. Par exemple `"(()())"` est bien parenthésée alors que `"())("` et `"(()"` ne le sont pas.

En utilisant un compteur qui compte le nombre de parenthèses ouvrantes que l'on a vu jusqu'à présent et qui ne sont pas fermées, écrire une fonction `bien_parenthesee(s: str) -> bool` nous indiquant si la chaine caractère `s`, que l'on supposera constituée uniquement de `'('` et de `')'`, est bien parenthésée.

Assertion, test unitaire

Exercice 5 : Chasse aux bugs

Les fonctions des questions suivantes sont buguées. Pour chaque fonction, le but est de proposer un test unitaire qui n'est pas passé par la fonction puis de corriger le bug.

1. On définit la suite de Fibonacci par

$$F_0 \coloneqq 0, \quad F_1 \coloneqq 1, \quad \text{et} \quad \forall n \in \mathbb{N}, \quad F_{n+2} \coloneqq F_{n+1} + F_n.$$

Donner un test unitaire qui n'est pas passé par la fonction suivante devant calculer le n-ième terme de la suite de Fibonacci

```python
def fibo(n):
    """fibo(n: int) -> int"""
    a = 0
    b = 1
    for _ in range(n - 1):
        a, b = b, a + b
    return b
```

puis corriger la fonction pour qu'elle devienne valide.

Sortie anticipée

2 Variable locale et globale

Variable locale

Variable globale

Composition de fonctions

3 Programmation récursive

Fonction récursive pure

Exercice 6 : Ensemble des parties

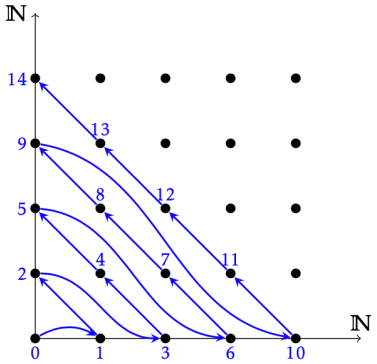
On suppose les ensembles représentés par des listes non triées d'éléments deux-à-deux distincts. Rédiger une fonction `subset` qui prend en argument un ensemble et qui renvoie l'ensemble de ses parties. Par exemple `subset([1,2,3])` doit renvoyer `[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]`.

Exercice 7 : Somme dans un arbre

Écrire une fonction `somme(t)` qui prend en paramètre une séquence imbriquée, de profondeur et de structure quelconque, dont tous les composants élémentaires sont des nombres, et qui calcule la somme de tous ces éléments. Par exemple `somme([[[1, 2], [3, 4, 5]], 6, [7, 8], 9])` devra renvoyer 45. Pour écrire cette fonction on pourra utiliser la fonction `isinstance(t, list)` qui permet de savoir si t est une liste.

Exercice 8 : Dénombrabilité de $\mathbb{N}^2$

On démontre que l'ensemble $\mathbb{N} \times \mathbb{N}$ est dénombrable en numérotant chaque couple $(x, y) \in \mathbb{N}^2$ suivant le procédé suggéré par la figure ci-dessous.



1. Rédiger une fonction récursive qui renvoie le numéro du point de coordonnées $(x, y) \in \mathbb{N}^2$.
2. Rédiger la fonction réciproque, là encore, de façon récursive.

Fonction récursive impérative

Exercice 9 : Triangles

1. Écrire une fonction récursive `triangle(n)` affichant un triangle de la manière suivante.

```
In [1]: triangle(5)
(*@\textcolor{purple}{*}@*)
(*@\textcolor{purple}{**}@*)
(*@\textcolor{purple}{***}@*)
(*@\textcolor{purple}{****}@*)
(*@\textcolor{purple}{*****}@*)
```

2. Écrire une fonction récursive `triangle_inverse(n)` affichant un triangle de la manière suivante.

```
In [2]: triangle_inverse(5)
(*@\textcolor{purple}{*****}@*)
(*@\textcolor{purple}{****}@*)
(*@\textcolor{purple}{***}@*)
(*@\textcolor{purple}{**}@*)
(*@\textcolor{purple}{*}@*)
```

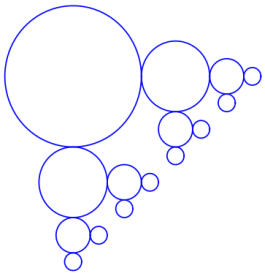
3. Écrire une fonction récursive `sablier(n)` affichant un demi-sablier de la manière suivante.

```
In [3]: sablier(4)
(*@\textcolor{purple}{****}@*)
(*@\textcolor{purple}{***}@*)
(*@\textcolor{purple}{**}@*)
(*@\textcolor{purple}{*}@*)
(*@\textcolor{purple}{**}@*)
(*@\textcolor{purple}{***}@*)
(*@\textcolor{purple}{****}@*)
```

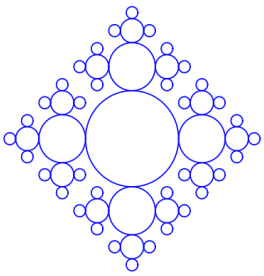
Exercice 10 : Cercles

On suppose disposer d'une fonction `circle([x, y], r)` qui trace à l'écran un cercle de centre (x, y) et de rayon r.

1. Définir une fonction récursive permettant de tracer le dessin ci-dessous où le cercle le plus gros est de rayon 1 de centre de coordonnées $(0, 0)$ et chaque cercle est de rayon deux fois plus petit que celui de la génération précédente.



2. Même question avec ce dessin.

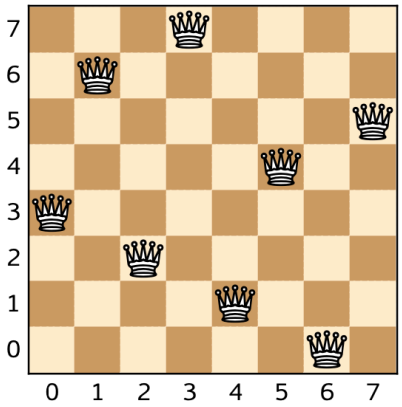


Exercice 11 : Variante sur les tours de Hanoï

Résoudre le problème des tours de Hanoï en s'imposant une contrainte supplémentaire : tout mouvement entre les tiges 1 et 3 est interdit

Exercice 12 : Problème des n reines

Le problème des n reines consiste à placer n reines sur un échiquier de sorte que deux reines quelconques ne puissent pas s'attaquer, c'est-à-dire qu'il ne faut pas que deux reines partagent une même ligne, une même colonne ou une même diagonale. De telles positions seront qualifiées dans cet exercice de *valides*. Il est évident que dans chaque colonne doit se trouver une et une seule reine. Ainsi, il est possible de représenter ce problème par un tableau de n cases $q = [q_0, \ldots, q_{n-1}]$ dans lequel q_j désigne la ligne dans laquelle est placée la reine de la colonne j. Par exemple, le dessin ci-dessous représente la solution $[3, 6, 2, 7, 1, 4, 0, 5]$.



On appellera solution partielle de rang j un tableau q de longueur n dont les j premières cases sont remplies avec des positions valides pour les reines, les $n - j$ autres cases restant à remplir.

Écrire une fonction `reine(q, j)` qui prend pour arguments un entier j et une solution partielle q et qui réalise les opérations suivantes.

— Si $j = n$, cette fonction se contente d'afficher le tableau q. Dans ce cas, le problème est résolu.
— Si $j < n$, cette fonction recherche parmi les n valeurs possibles pour q_j celles qui correspondent à des positions valides et pour chacune d'elles poursuit la recherche au rang $j + 1$.

Fonctions mutuellement récursives