

## 💫 Terminale Spé NSI : Évaluation en classe n°3 💫

### EXERCICE 1

On donne ci-dessous le code d'une fonction récursive écrite en langage Python :

```

1 def chaineStr(n : int) -> str:
2     if n > 1:
3         return chaineStr(n - 1) + n * "-" + chaineStr(n - 2)
4     else:
5         return (n + 1) * "+"

```

1. Que renvoie cette fonction pour l'exécution `chaineStr(0)` ? Expliquez pourquoi.
2. Que renvoie cette fonction pour l'exécution `chaineStr(2)` ? Expliquez pourquoi.
3. Représentez l'arbre des appels récursifs pour l'exécution `chaineStr(4)` en respectant les trois consignes suivantes :
  - 👉 Numérotez les appels descendants avec une couleur.
  - 👉 Numérotez les renvois avec une autre couleur.
  - 👉 Indiquez les valeurs renvoyées dans une troisième couleur.
4. Déduisez de cet arbre la valeur renvoyée par l'appel `chaineStr(4)`.

### EXERCICE 2

L'objectif de cet exercice est d'implémenter plusieurs stratégies de calcul de la puissance entière  $n$  d'un nombre réel  $x$  non nul, soit  $x^n$  comme  $3.2^2$  par exemple, mais sans utiliser l'opération `**` du langage Python bien sûr.

On rappelle que pour tout nombre réel non nul désigné par la lettre  $x$ , on a  $x^0 = 1$ .

1. Une version itérative.

On donne la signature et la spécification d'une fonction nommée `puissance1`.

```

1 def puissance1(x : float, n : int) -> float:
2     """Cette fonction renvoie la valeur de x à la puissance n.
3     Elle est itérative.
4     x doit être un flottant ou un entier non nul.
5     n est un entier naturel.
6     """
7     # pré-conditions
8     .....
9     .....
10    # traitement
11    acc = .....
12    for .....:
13        acc = .....
14    return .....

```

- (a) Écrivez en lignes 8 et 9 les deux assertions qui permettent de vérifier les deux pré-conditions.
- (b) Complétez les lignes 11, 12, 13 et 14 afin que le code réalise la spécification de cette fonction.
- (c) Écrivez une série de 6 appels de cette fonction qui testent bien le code proposé y compris ses assertions.

## 2. Une première version récursive.

L'algorithme s'appuie sur la définition mathématique suivante

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ x^{n-1} \times x & \text{si } n \geq 1 \end{cases}$$

On donne la signature et la spécification d'une fonction nommée `puissance2` qui doit constituer une implémentation de cet algorithme.

```

1 def puissance2(x : float, n : int) -> float:
2     """Cette fonction renvoie la valeur de x à la puissance n.
3     Elle est récursive.
4     x doit être un flottant ou un entier non nul.
5     n est un entier naturel.
6     """
7     # pré-conditions
8     .....
9     .....
10    # moteur récursif
11    def puiss_rec(.....) -> float:
12        if .....:
13            return .....
14        else:
15            return .....
16    # appel récursif principal
17    return .....
```

- Complétez les pré-conditions aux lignes 8 et 9 puis le reste du code aux lignes 11, 12, 13, 15 et 17 afin que cette fonction mette en œuvre l'algorithme proposé.
- Commentez les lignes 13 et 15.
- Expliquez pourquoi cette récursivité n'est pas terminale.

## 3. Une seconde version récursive.

```

1 def puissance3(x : float, n : int) -> float:
2     """Cette fonction renvoie la valeur de x à la puissance n.
3     Elle est récursive terminale.
4     x doit être un flottant ou un entier non nul.
5     n est un entier naturel.
6     """
7     # pré-conditions
8     .....
9     .....
10    # moteur récursif
11    def puiss_rec(.....) -> float:
12        if .....:
13            return .....
14        else:
15            return .....
16    # appel récursif principal
17    return .....
```

Dans cette version, vous devez introduire un accumulateur afin de rendre la récursivité terminale. Complétez le code de la fonction `puissance3` qui doit réaliser cette demande.

#### 4. Un autre algorithme récursif.

L'algorithme de cette dernière version prend appui sur la propriété mathématique suivante

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ (x^{n/2})^2 & \text{si } n \text{ est pair} \\ x \times (x^{n/2})^2 & \text{si } n \text{ est impair} \end{cases}$$

où  $n/2$  désigne le quotient de la division euclidienne de  $n$  par 2.

- (a) Expliquez pourquoi cette écriture permet de mettre en place une stratégie « diviser pour régner ».

En particulier, situez précisément les trois étapes **diviser**, **régner** et **combinaison**.

- (b) On donne la signature et la spécification d'une fonction nommée `_expo_rec` qui ne réalise que le calcul de  $x$  à la puissance  $n$  en mettant en œuvre cette stratégie.

```

1 def _expo_rec(x : float, n : int) -> float:
2     """Cette fonction renvoie la valeur de x à la puissance n.
3     Elle est récursive.
4     """
5     # traitement
6     if n == 0:
7         return .....
8     else:
9         # .....
10        .....
11        # .....
12        .....
13        # .....
14        if .....:
15            return .....
16        else:
17            return .....
```

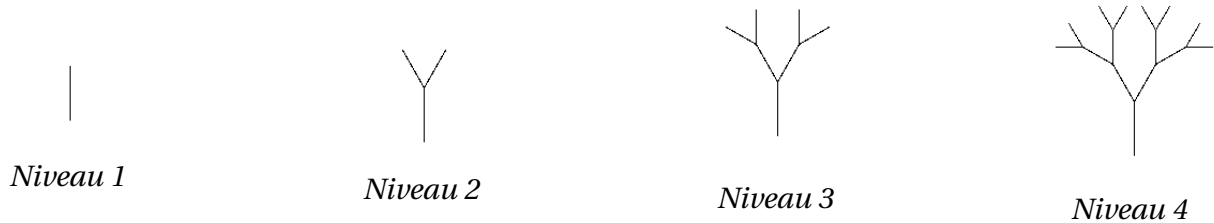
- Complétez les lignes de code 7, 10, 12, 14, 15 et 17 en utilisant le produit avec `*` plutôt que le carré avec `**2` afin de demeurer dans l'esprit de l'exercice.
- Complétez les lignes de commentaires 9, 11 et 13.
- Expliquez pourquoi le nom de cette fonction est préfixé du signe *underscore*.

Instruction	Explication
<code>divmod(a, b)</code>	Renvoie le quotient et le reste de la division euclidienne de $a$ par $b$ qui sont deux nombres entiers naturels. Exemple : <code>divmod(7, 2)</code> renvoie les deux entiers 3 et 1.
<code>q, r = divmod(a, b)</code>	Affecte le quotient et le reste de la division euclidienne de $a$ par $b$ aux variables nommées <code>q</code> , pour le quotient, et <code>r</code> , pour le reste. Exemple : <code>a, b = divmod(7, 2)</code> affecte la valeur 3 à <code>a</code> et 1 à <code>b</code> .

**EXERCICE 3**

On souhaite coder le dessin d'un arbre fractal en Y en utilisant le module `turtle`.

On donne ci-dessous les dessins obtenus pour les quatre premiers niveaux :



Les niveaux désignent le nombre « d'étages » de l'arbre. La longueur du tronc au niveau 1 est fixée puis, cette longueur est réduite de 20% pour chaque étage supplémentaire.

On rappelle qu'il suffit de multiplier une grandeur par 0,8 pour la réduire de 20%.

On fournit une partie du code du module.

```

1  import turtle as trt
2
3  # constantes
4  angle : int = 30
5  taille : float = 50.0
6
7  # fonction interface
8  def arbreEnY(nbNiv : int) -> None:
9      """Cette fonction construit un arbre fractal en Y sur nbNiv niveaux.
10         L'arbre de niveau 1 a la taille initiale, la taille
11         est réduite de 20% à chaque niveau supérieur.
12     """
13     # pré-conditions
14     assert isinstance(nbNiv, int) and nbNiv >= 0
15     # traitement
16     _dessineArbre(nbNiv, taille)
17     return None
18
19
20 # fonction récursive
21 def _dessineArbre(nbNiv : int, longueur : float) -> None:
22     """Cette fonction construit récursivement un arbre fractal
23         en Y sur nbNiv niveaux.
24         longueur est la taille du tronc.
25     """

```

La constante `angle` fixe la valeur de l'inclinaison d'une des deux branches du Y vers la droite ou vers la gauche par rapport à la direction courante.

La constante `taille` fixe la longueur du tronc pour l'arbre de niveau 1.

Initialement la tortue est orientée verticalement vers le haut et le stylo est posé pour dessiner.

On rappelle ci-dessous les instructions du module `turtle` utiles pour la suite.

Instruction	Description
<code>forward(longueur)</code>	La tortue avance de longueur pixels.
<code>backward(longueur)</code>	La tortue recule de longueur pixels.
<code>right(angle)</code>	La tortue s'oriente de angle degrés vers la droite par rapport à sa direction actuelle.
<code>left(angle)</code>	La tortue s'oriente de angle degré vers la gauche par rapport à sa direction actuelle.






1. Sur chacun des dessins des niveaux 2, 3 et 4, identifiez les dessins du niveau précédent en les entourant.
2. Écrivez un algorithme ou le code Python de la fonction `_dessineArbre`.

Nom :

Prénom :

\* \* \* \* \*

01/12/2023 - Récursivité - Diviser pour régner

Réf	Intitulé	Code
Algorithmique et programmation	Décomposer et recomposer	
Algorithmique et programmation	Anticiper et tester	
Algorithmique et programmation	Évaluer un script	
Algorithmique et programmation	Décrire et spécifier	
Communiquer à l'écrit	Développer une argumentation	
Communiquer à l'écrit	S'exprimer avec clarté et précision	